Razvoj okoline za učinkovito stvaranje složenog sustava SQL upita

Radić, Svetozar

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

Diplomski rad

RAZVOJ OKOLINE ZA UČINKOVITO STVARANJE SLOŽENOG SUSTAVA SQL UPITA

Svetozar Radić

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 21.02.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Svetozar Radić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1010R, 09.10.2018.
OIB studenta:	00855817623
Mentor:	Prof.dr.sc. Goran Martinović
Sumentor:	
Sumentor iz tvrtke:	Jonathan van Driessen
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 1:	Prof.dr.sc. Goran Martinović
Član Povjerenstva 2:	Izv. prof. dr. sc. Krešimir Nenadić
Naslov diplomskog rada:	Razvoj okoline za učinkovito stvaranje složenog sustava SQL upita
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U ovom diplomskom radu potrebno je najprije opisati zahtjevnost stvaranja složenih sustava SQL upita za različite primjene, a s posebnim naglaskom na poslovne i promidžbene primjene. Nadalje, treba analizirati zahtjeve i predložiti model okoline za učinkovito stvaranje takvih sustava upita po načelu drag and drop, opisati potrebne programske jezike, tehnologije i razvojne okoline za agilni razvoj, nadzor pogrešaka u stvarnom vremenu i puštanje programskog rješenja u upotrebu. Također, treba opisati i pri razvoju sustava koristiti naprednije postupke rukovanja podacima kao što su podupiti, pametni filtri podataka, ekstenzije i relacije među podacima. U praktičnom dijelu rada, potrebno je programski ostvariti navedenu
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	21.02.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 16.03.2021.

Ime i prezime studenta:	Svetozar Radić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1010R, 09.10.2018.
Turnitin podudaranje [%]:	12

Ovom izjavom izjavljujem da je rad pod nazivom**: Razvoj okoline za učinkovito stvaranje složenog sustava SQL upita**

izrađen pod vodstvom mentora Prof.dr.sc. Goran Martinović

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD

Ideja za stvaranje okoline opisane u ovom radu proizašla je iz problema pripreme odnosno manipulacije podatcima koji služe za stvaranje marketinške promidžbe [1]. Marketinška promidžba ima svrhu približiti, odnosno zainteresirati određenu skupinu ljudi za neki proizvod ili usluge određene organizacije preko raznih medija za oglašavanje. S obzirom na to da se marketinškom promidžbom želi obuhvatiti samo ciljne skupine ljudi, podaci korišteni u promidžbi trebaju se prilagoditi takvoj potrebi. Prikupljeni podaci o klijentima kao što je raspon starosti, lokacija, grana industrije itd. pohranjuju se u formate za rad s podacima, npr. Excel tablice [2]. Nakon što su podaci prikupljeni, potrebno je iz njih izdvojiti one podatke pomoću kojih bi se promidžba što bolje odnosila na klijente kojima se želi pristupiti. Postupak prilagodbe podataka za svrhu promidžbe predstavlja vrlo složen i dugotrajan proces u kojemu se analiziraju, filtriraju i obrađuju podaci koji će se koristiti u daljnjim procesima. Takve podatke potrebno je izdvojiti iz ogromne količine podataka korištenjem Structured Query Language (SQL) upita. No, tu se mogu pojaviti razni problemi budući da je cijeli postupak pripreme podataka za promidžbu dinamičan proces gdje se podaci moraju stalno ažurirati [3].

Cilj diplomskog rada je izrada okoline za brže, jednostavnije i učinkovitije stvaranje SQL upita kojima će se pojednostaviti cijeli postupak pripreme podataka za marketinšku promidžbu. Stvaranje SQL upita ostvarit će pomoću tri glavna koraka. U prvom koraku definirat će se izvorišna skupina podataka te filtri koji će omogućiti izdvajanje potrebnih podataka. U drugom koraku definirat će se odredište gdje će se dobiveni podaci spremiti. Konačno, u trećem koraku korisniku će se prikazati pregled dobivenih podataka. Upravljanje okolinom bit će ostvareno preko okoline u oblaku računala, dok će upravljanje podacima biti napravljeno preko baze podataka.

Na početku drugog poglavlja opisana je okolina Salesforce Marketing Cloud te SQL kao jezik za rukovanje podacima. Također, prikazuje se i neki od postojećih rješenja za pomoć pri radu sa SQL – om. U trećem poglavlju dan je model programskog rješenja za kreiranje SQL upita pomoću metode „*drag and drop*“. U četvrtom poglavlju opisano je programsko rješenje te alati, računalne platforme i tehnologije korištene za izradu cjelokupne programske podrške koji su popraćeni odgovarajućim kodovima za pojedina rješenja. U petom poglavlju opisuju se značajke i način rada cjelokupnog sustava, prikazani su pojedini dijelovi sučelja, te ispitan i analiziran način rada sustava.

2. PREGLED PROBLEMATIKE STVARANJA SQL UPITA I POSTOJEĆI ALATI.

U ovom poglavlju opisat će se Salesforce Marketing Cloud okolina te problemi i rješenja za ostvarivanje segmentacije, te SQL jezik i neka prijašnja programska rješenja vezana za korištenje SQL jezika.

2.1. Okolina Salesforce Marketing Cloud

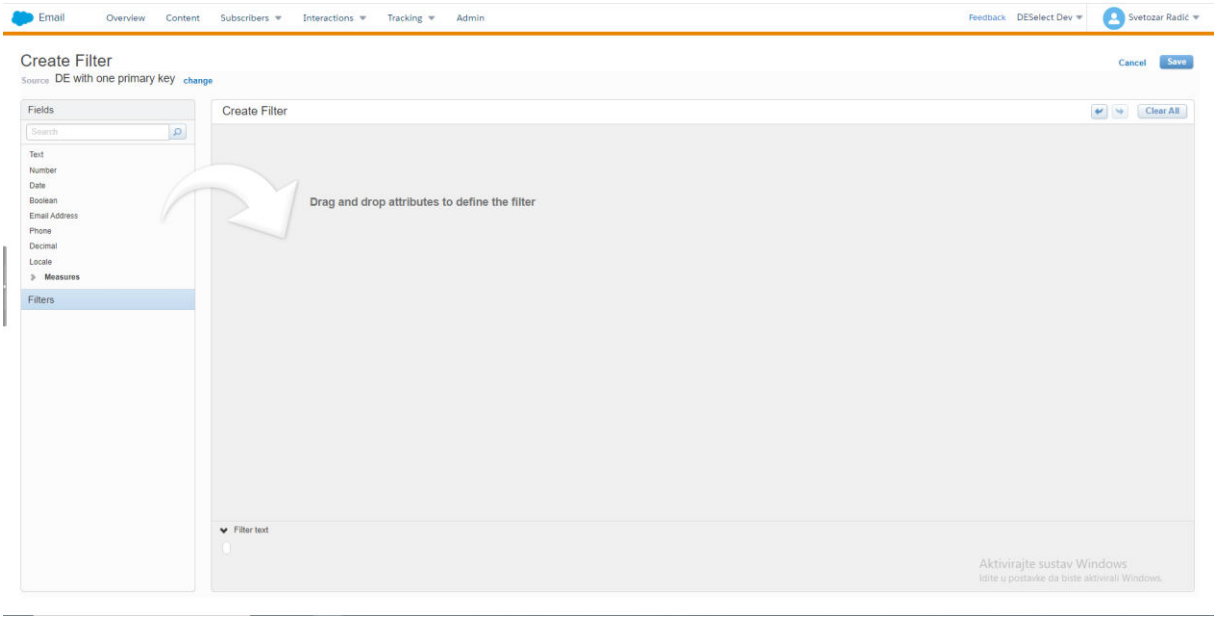
Salesforce Marketing Cloud jedan je od vodećih višenamjenskih marketinških platformi za stvaranje i upravljanje uspješnim marketinškim promidžbama i njegovanja odnosa s kupcima [4]. Odnos s kupcima predstavlja temeljni pojam u marketingu bilo da je riječ o B2B (Business-to-Business) ili B2C (Business-to-Consumer) tipu poslovanja. B2B (Business-to-Business) predstavlja posao usmjeren poslovanju što znači da B2B tvrtke prodaju proizvode ili usluge drugim tvrtkama. Primjer jedne takve tvrtke je tvrtka Maersk Line koja je svjetska poznata B2B tvrtke s više od 600 brodova u svojoj floti, a osnovu njihovih klijenata čine tvrtke koje se bave uvozom i izvozom. S druge strane B2C (Business-to-Consumer) predstavlja posao usmjeren prema potrošaču što znači da B2C tvrtke prodaju proizvod ili usluge kupcima za osobnu upotrebu, poput usluge putovanja, odjeće, automobile i slično [5]. Glavni cilj marketinških promidžbi je stvaranje promidžbi koje će se što bolje odnositi na kupce koji se ciljaju. Salesforce Marketing Cloud pruža razne usluge kojima će se taj cilj postići. U vezi s navedenim, javljaju se mnogi problemi posebno problemi vezani uz rukovanje korisničkim podacima te njihovo korištenje pri kreiranju ciljanih marketinških promidžbi [6]. Podaci u Salesforce Marketing Cloudu predstavljaju bilo koje podatke koji se mogu prikupiti (podaci o pretplatnicima, analitički podaci, opisni podaci, itd..) te se takvi podaci pohranjuju u tablice koje nazivamo podatkovna proširenja (engl. data extensions). Podatkovna proširenja mogu se filtrirati, pretraživati i uređivati kako bi se kreirale što preciznije promidžbene poruke. Za slanje promidžbe mora se pripremiti određena publika. Konkretno, moraju se stvoriti proširenja podataka s podacima o ciljanim kontaktima te dodatnim podacima koji se koriste za personalizaciju.

2.2.1. Način za segmentiranje u Salesforce Marketing Cloudu

Uobičajeni načini pripreme publike za promidžbe u Salesforce Marketing Cloudu su:

Filtri [7] : Filtri su relativno jednostavan način filtriranja proširenja podataka. Međutim, ne mogu se pouzdano koristiti u više proširenja podataka i njihove su funkcije ograničene.

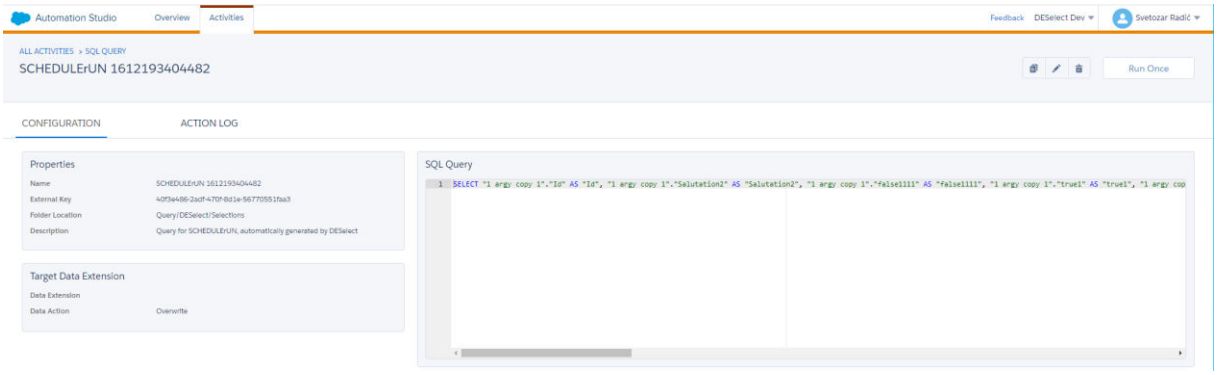
Također ni korisničko iskustvo nije sjajno. Prikaz sučelja za korištenje filtra u Salesforce Marketing Cloudu prikazano je na slici 2.1.



Sl. 2.1. Prikaz filtra u Salesforce Marketing Cloud

Salesforce Izvješća [8] Pod pretpostavkom da je Salesforce CRM(Customer Relationship Management) spojen s SFMC-om (na primjer, Service Cloud), Salesforce izvješća pružaju više funkcionalnosti u odnosu na filtre u Salesforce Marketing Cloudu, ali i dalje postoje stroga ograničenja. CRM podaci mogu se koristiti samo u Salesforce izvješćima, te se ne mogu koristiti podaci o angažmanu promidžbe zabilježeni u Salesforce Marketing Cloudu. Također, financijski trošak je udvostručen na troškove licenci, jer trgovci sada koriste dvije platforme te je česta zamjena između te dvije platforme neučinkovita.

SQL upiti [9]: Kada se gleda iz uske perspektive funkcionalnosti, čini se da SQL upiti ispunjavaju većinu potreba za segmentacijom. Međutim, ovaj je pristup nezgrapan te je pisanje SQL upita zahtjevan i oduzima puno vremena. Većina timova nije spremna za hladnu SQL stvarnost kada započnu s Salesforce Marketing Cloudom i shvate da trebaju unaprijediti svoj tim ili vanjske resurse. Bilo koji put je spor i skup. Prikaz sučelja za stvaranje SQL upita prikazan je na slici 2.2.



Sl. 2.2. Prikaz SQL upita u Salesforce Marketing Cloud-u

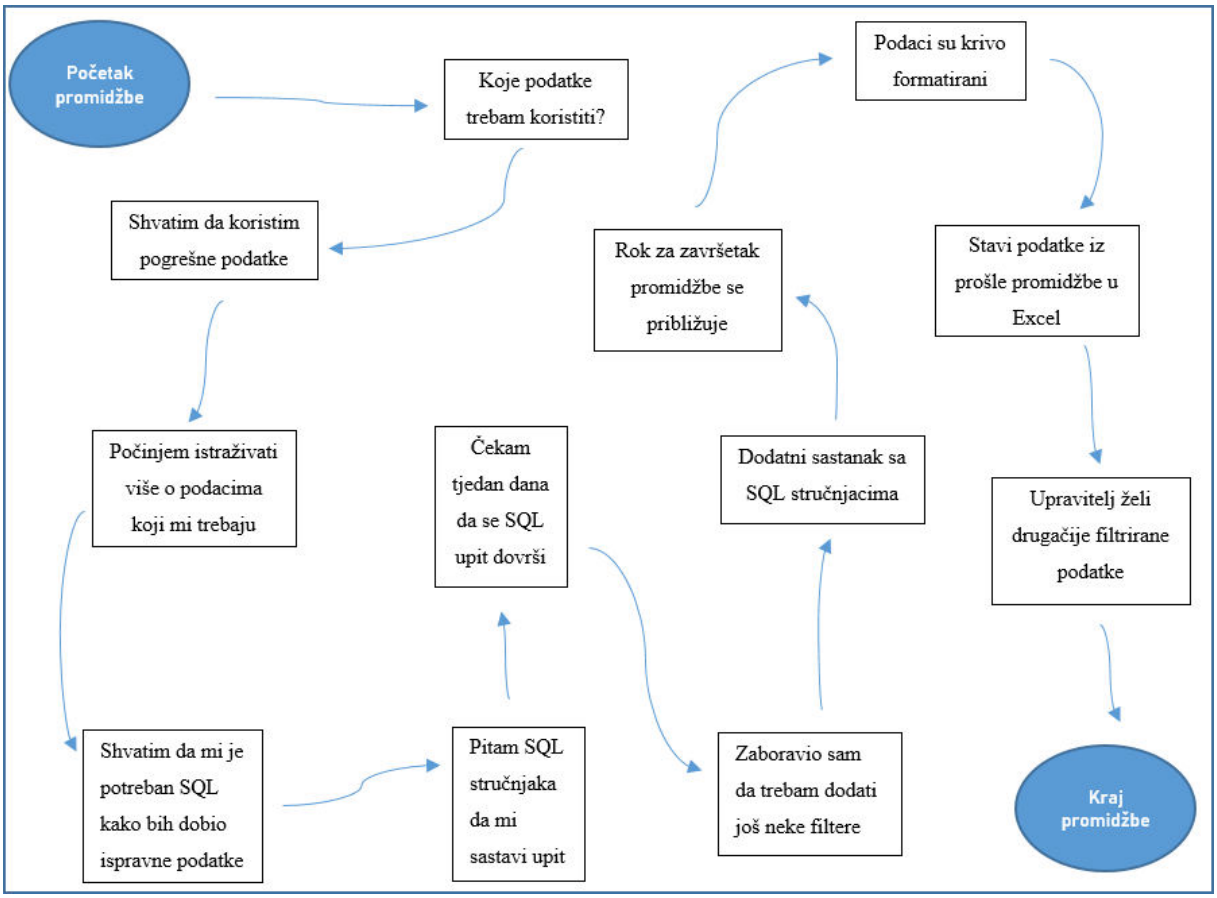
Rješenje za upravljanje publikom [10]: Glavna prednost rješenja za upravljanje publikom je njegova sposobnost na izvrstan način pregledavanja velikih skupova podataka. Međutim, takav pristup snosi znatne troškove i za provedbu rješenja potrebni su vam stručnjaci. Tijekom provedbe implementacije, marketinški tim također treba uložiti vrijeme za objašnjenje zahtjeva i validacije. Jednom implementirani podaci se obično kopiraju svakodnevno sa Salesforce Marketing Clouda, tako da se uvijek koriste podaci koji su zastarjeli. Ukoliko se trebaju koristiti različiti podaci ili istražiti podaci iz drugog kuta, stručnjaci moraju ponovno konfigurirati rješenje koje će se naplatiti.

Iz navedenog, nijedan od tih pristupa nije osobito sjajan. Stvaranje proširenja podataka dugotrajno je, uklanjanje pogrešaka SQL upita zahtjevno, te i nakon svega toga postoji mogućnost za potrebu korištenja Excela za uređivanje podataka što ne predstavlja nimalo lak postupak uređivanja podataka.

2.2.2. Problemi prilikom segmentiranja u Salesforce Marketing Cloud okolini

Prilikom segmentiranja podataka unutar Salesforce Marketing Clouda koriste se razni SQL upiti koji mogu biti vrlo složeni. Tijekom procesa segmentiranja odnosno pisanja SQL upita u svrhu dohvaćanja i pripreme ispravnih podataka često dolazi do tzv. ping pong problema. Na početku marketinški stručnjaci trebaju odabrati koji će se podaci koristiti prilikom segmentiranja. U tom trenutku shvate da im je potreban SQL upit kako bi dobili željene podatke. S obzirom da marketinški stručnjaci ne posjeduju znanje o SQL jeziku, ukoliko žele dobiti ispravne podatke, moraju se obratiti SQL stručnjaku za sastavljanje SQL upita. Sastavljanje SQL upita u ovakvim okolnostima može trajati nekoliko dana. Prilikom završetka SQL upita, marketinški stručnjak može uvidjeti kako su potrebni dodatni filtri za stvaranje SQL upita. Cijeli proces stvaranja SQL upita od strane SQL stručnjaka ponovo se pokreće što oduzima još nekoliko dana do početka roka za završetak promidžbe. Nakon

nekoliko takvih iteracija dobiva se željeni SQL upit te su željeni podaci zadovoljavajući, ali se sada javljaju drugi problemi. Nakon dobivanja željenih podataka, postoji mogućnost da podaci nisu dobro formatirani ili upravitelj želi drugačije filtrirane podatke, te se cijeli postupak ponovo pokreće, što uzrokuje kašnjenje s završetkom promidžbe. Cjelokupni postupak tzv. ping pong problema prilikom segmentiranja podataka unutar Salesforce Marketing Clouda prikazan je na slici 2.3.



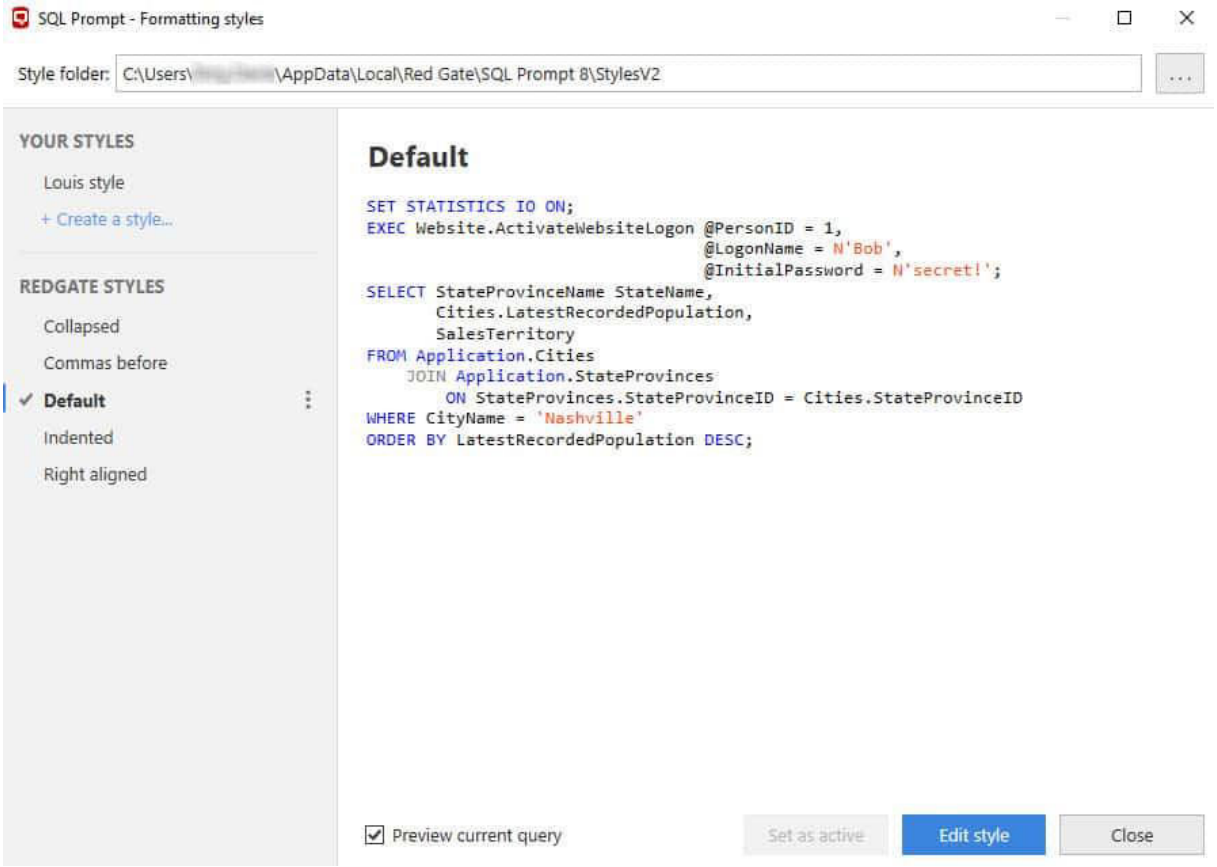
Sl. 2.3. Prikaz tzv. ping pong problema prilikom stvaranja promidžbe

Ovaj problem može produljiti očekivan rok za završetak promidžbe što tvrtki uzrokuje gubitak dragocjenog vremena i novca. Zbog toga je trebalo osmisliti rješenje koje će ubrzati i olakšati pripremu podataka za segmentiranje i ciljanje korisnika.

2.2. SQL jezik i postojeći alati za stvaranje SQL upita

Kako bi se iz mnoštva podataka izdvojili samo oni koji su potrebni koristi se jezik SQL. SQL je strukturni jezik za postavljanje upita nad bazama podataka. Razvio ga je IBM 70-ih godina prošlog stoljeća. Postoje brojna rješenja za rad sa SQL upitima. Nekih od postojećih rješenje prema [11] prikazana su u nastavku.

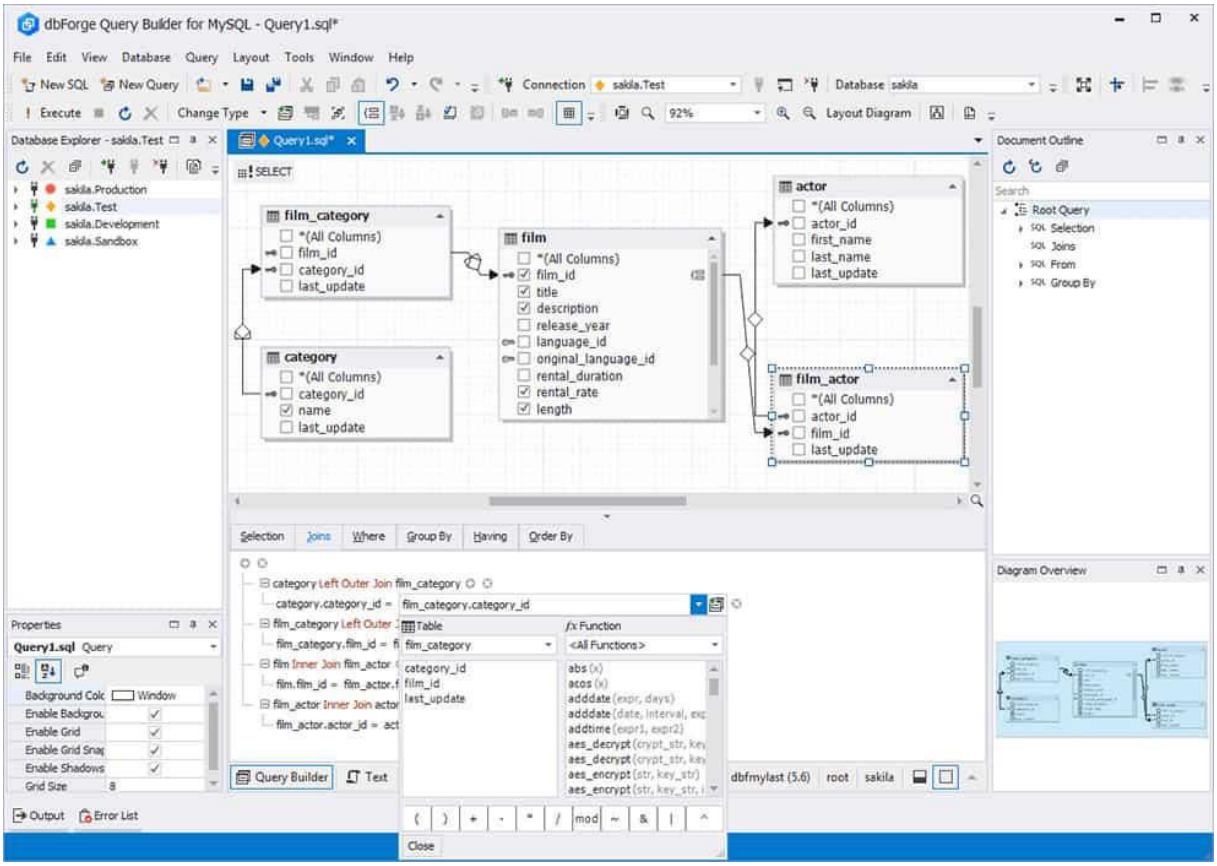
Redgate SQL Prompt SQL uređivač s kontekstualnim prediktivnim tekstom koji može predložiti sljedeću ključnu riječ koja je potrebna za stvaranje SQL upita. Prikaz sučelja Redgate SQL Prompt alata prikazan je na slici 2.4.



Sl. 2.4. Prikaz sučelja alata Redgate SQL Prompt

Redgate SQL Prompt je alat za stvaranje SQL-a koji može dati prijedloge kodova prilikom tipkanja. Programska podrška daje kontekstualne preporuke na temelju pravila analize koda, a postoji i biblioteka isječaka koda na koju se korisnik može pozivati. Postoje i opsežne mogućnosti oblikovanja, tako da se mogu odabrati skripte koje se žele formatirati ili blokirati formatiranje određenih blokova. Iako je ovaj alat vrlo koristan još uvijek je potrebno znanje pisanja SQL upita.

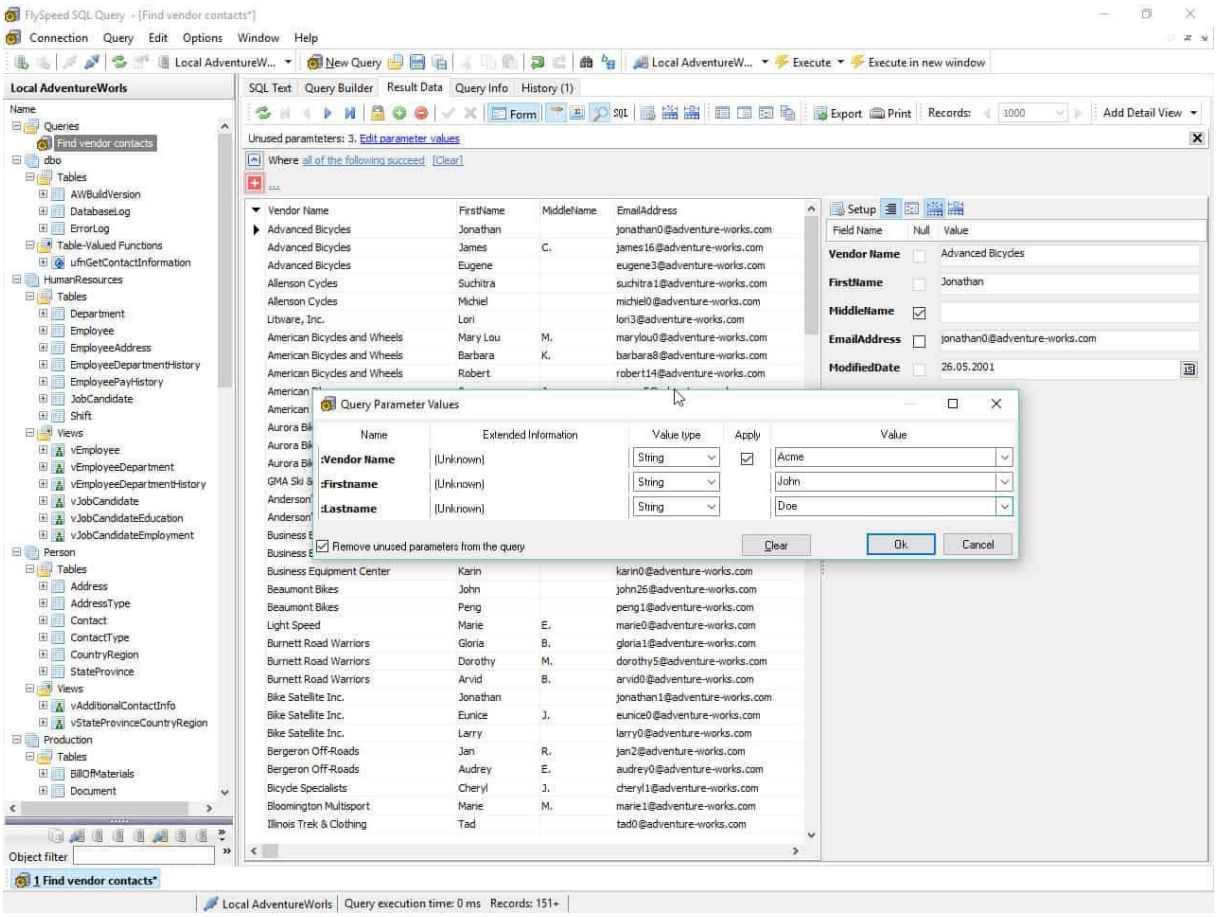
Sljedeće postojeće rješenje je dbForge Query Builder - vizualni graditelj SQL-a koji uključuje elemente za odabir te metodu „*drag and drop*“ kako bi pomogao u stvaranju upita. Prikaz sučelja alata dbForge Query Builder prikazan je na slici 2.5.



Sl. 2.5. Prikaz sučelja alata dbForge Query Builder

dbForge Query Builder, graditelj upita, dizajniran je kako bi pomogao u stvaranju složenih SQL upita. Upiti se mogu crtati kroz dijagram vizualnih upita te se mogu dodavati pod-upiti koji će se izgraditi na temeljima glavnog upita. Tu je i značajka „*drag and drop*“ pomoću koje se tablice mogu lako dodavati. Kako bi pomogao u uređivanju koda, dbForge Query Builder ima automatsku provjeru SQL sintakse. Također postoji mogućnost korištenja općenitijih značajki uređivanja SQL-a poput oznaka, pretraživanja teksta i bojanja. Nakon dovršetka kodiranja podaci se mogu izvoziti u 10 različitih formata, uključujući HTML, CSV, XML, PDF, MS Excel, MS Access, DBF, ODBC i Tekst.

FlySpeed SQL Query - SQL graditelj koji ima vizualne značajke „*drag and drop*“, kao i provjeru pravopisa i prijedloge za pisane SQL upita. Prikaz sučelja alata FlySpeed SQL Query prikazan je na slici 2.6.



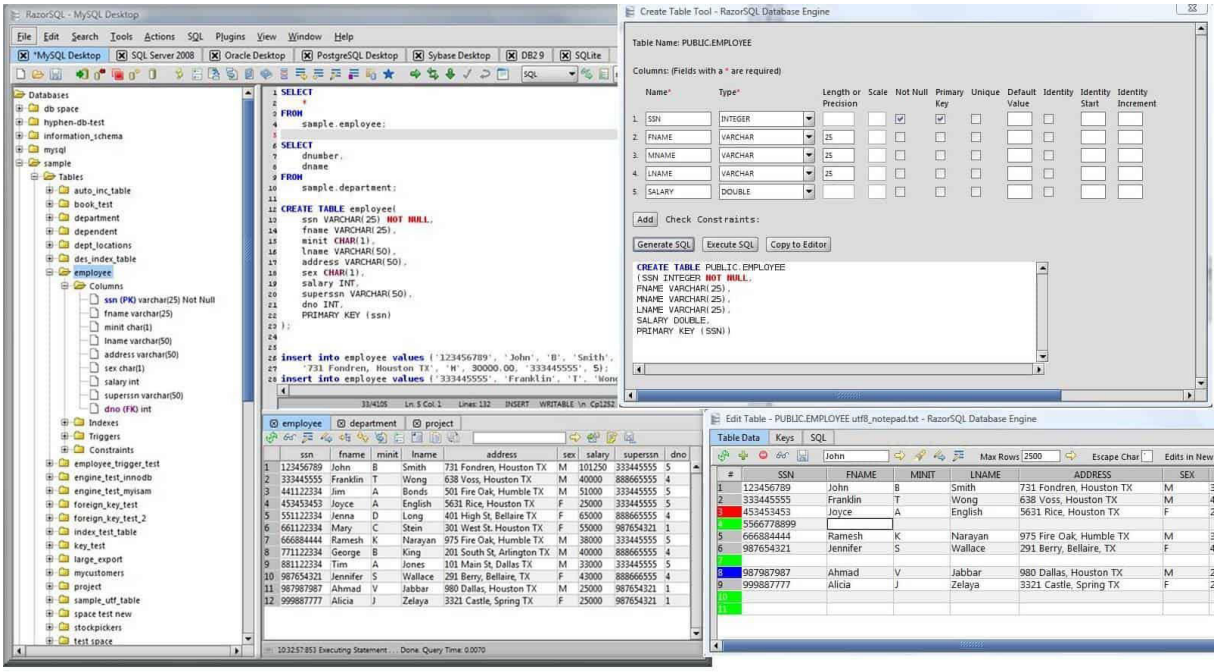
Sl. 2.6. Prikaz sučelja alata FlySpeed SQL Query

FlySpeed SQL Query je graditelj upita za Windows OS koji se može koristiti za stvaranje SQL upita putem metode „*drag and drop*“ i pomoću alata za izgradnju vizualnih upita. Omogućuje složene upite i uređivanje pod-upita u vizualnim i SQL tekstualnim načinima. Formatiranje je prilagodljivo tako da se može odlučiti kako će kôd biti predstavljen.

Da bi se poboljšalo iskustvo kodiranja, SQL uređivač teksta nudi dovršavanje koda s čime se postiže brža izrada koda. Tu je i sintaksa koja se ističe kako bi se olakšala navigacija. Također se redovito pohranjuje i povijest izvršavanja upita kako bi se na kraju sesije moglo pronaći mjesto na kojem je korisnik stao.

Programska podrška podržava SQL sintaksu za alate kao što su Microsoft SQL Server, MySQL, PostgreSQL, Oracle, InterBase, Firebird, MS Access, MS Excel, SQLite, Advantage DB i još mnogo toga. Također je višenitni što znači da se svaki upit izvršava na drugoj niti.

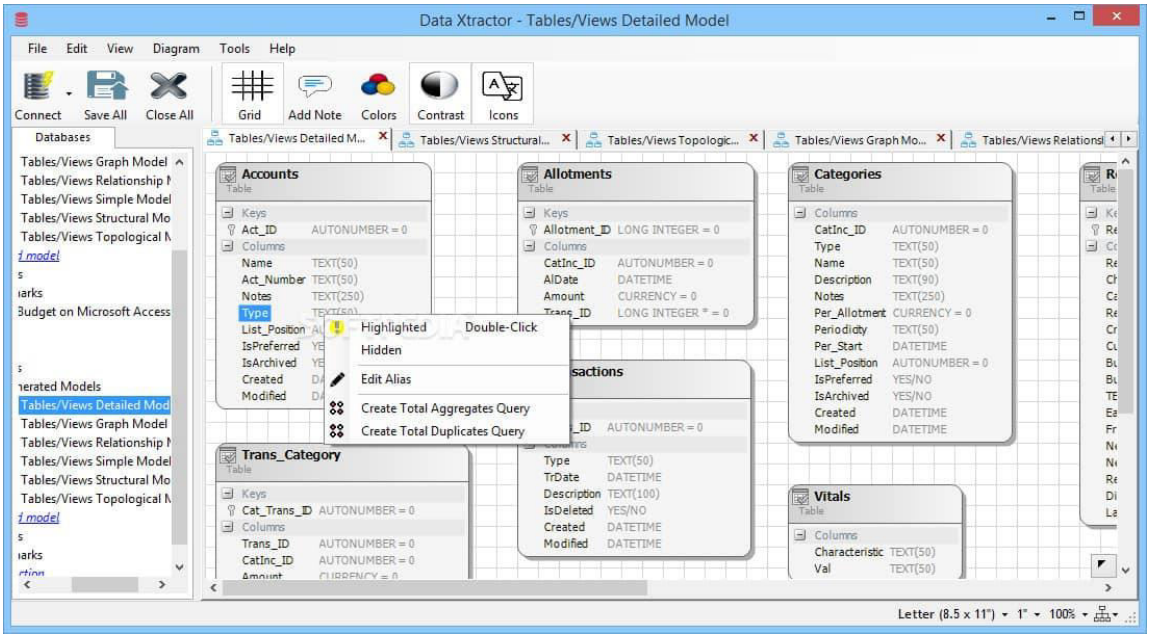
RazorSQL je vizualni graditelj SQL upita koji uključuje kodiranje sintakse u boji i podudaranje zagrada kako bi se izbjegle uobičajene pogreške. Prikaz sučelja alata RazorSQL prikazan je na slici 2.7.



Sl. 2.7. Prikaz sučelja alata RazorSQL

RazorSQL ima SQL alat za izradu upita koji korisniku omogućava vizualno sastavljanje upita. Može se odabrati vrsta SQL izraza koji se želi generirati odabirom stupaca i operacija koje treba uključiti. Alat je jednostavan za kretanje s četiri kartice za unošenje promjena u izjavama; odaberite, umetnite, ažurirajte i izbrišite.

Data Xtractor je vizualni graditelj SQL-a namijenjen onima koji nemaju iskustvo u pisanju SQL upita. Prikaz sučelja alata Data Xtractor prikazan je na slici 2.8.



Sl. 2.8. Prikaz sučelja alata Data Xtractor

Data Xtractor je SQL alat za upite koji korisnicima omogućuje stvaranje upita bez znanja o SQL-u. Data Xtractor dolazi s vizualnim graditeljem SQL upita pod nazivom Query Xtractor. Query Xtractor može stvarati SQL upite samo za čitanje te podržava baze podataka, uključujući MySQL, PostgreSQL, Oracle, SQL Server, Amazon Redshift, SQLite i Azure. Nakon što se izvrši upit, rezultati su prikazani u obliku proračunske tablice. Generator SQL upita automatski generira SQL upite za pojedine dobavljače. Također nije potrebno ništa instalirati za generiranje koda za drugu platformu što ga čini učinkovitim za stvaranje SQL upita.

Svi navedeni alati za kreiranje SQL upita pružaju razne mogućnosti, sučelja i biblioteke za kreiranje upita ali korisnici i dalje moraju imati znanje o SQL- u. Također, neki alati pružaju korisničko sučelje koje nije jednostavno koristiti odnosno korisničko iskustvo nije na visokoj razini.

3. MODEL OKOLINE ZA UČINKOVITO STVARANJE SLOŽENOG SUSTAVA SQL UPITA

U ovom poglavlju na osnovu prethodnih poglavlja uspostavit će se konačni model okoline za učinkovito stvaranje složenog sustava SQL upita sa svim njezinim koracima, preko definiranja izvorišnog skupa podataka i filtera, definiranja ciljnog podatkovnog proširenja te krajnjim tabličnim prikazom podataka dobivenih generiranim SQL upitom.

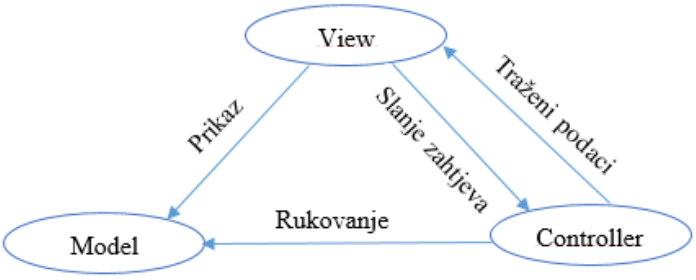
3.1. Opis modela okoline

Jedan od neizostavnih uvjeta za korištenje ove okoline je taj da korisnik mora imati Salesforce Marketing Cloud korisnički račun, budući da je okolina namijenjena za korištenje unutar Salesforce Marketing Clouda. Nakon što se korisnik prijavi u Salesforce Marketing Cloud potrebno je, najprije, instalirati odgovarajući paket kako bi okolina bila dostupna za korištenje. Paket, koji je potrebno instalirati, će sadržavati dvije komponente. Prva komponenta je komponenta Marketing Cloud App koja omogućuje prikazivanje korisničkog sučelja okoline unutar Salesforce Marketing Clouda. Na ovaj se način marketinškim stručnjacima omogućavaju dodatne funkcionalnosti unutar platforme Salesforce Marketing Cloud. Druga potrebna komponenta paketa je komponenta API Integration koja omogućava povezivanje prikazanog korisničkog sučelja sa poslužiteljskom stranom koja će se povezati sa Salesforce Marketing Cloudom te pomoću aplikacijsko programskih sučelja (engl. application programming interface, API) koristiti resurse Salesforce Marketing Clouda u svrhu pružanja podrške klijentskoj strani. API-ji predstavljaju sučelja kroz koja se odvijaju interakcije između davatelja usluge i aplikacije koja koristi resurse davatelja usluge. Nakon instaliranje potrebnih paketa potrebno je spremiti podatke o korisniku koji će se koristit za autentifikaciju pomoću standarda OAuth 2.0. Standard OAuth 2.0 pruža klijentima siguran ovlašten pristup resursima poslužitelja u ime vlasnika resursa. Određuje postupak za vlasnike resursa za odobravanje pristupa trećih strana njihovim resursima poslužitelja bez davanja vjerodajnica. Dizajniran je posebno za rad s protokolom za prijenos hiperteksta (HTTP) te u osnovi omogućuje odobrenje pristupnih tokena koje trećim stranama izdaje autorizacijski poslužitelj, uz odobrenje vlasnika resursa. Treća strana zatim koristi pristupni token za pristup zaštićenim resursima koje sadrži poslužitelj resursa [12]. Nakon uspješne autentifikacije, s obzirom da svaki korisnik pripada nekoj organizaciji, potrebno je spremiti informacije o organizaciji. Spremanje informacije o korisniku i organizaciji kojoj korisnik pripada ostvarit će se pomoću poslužiteljske strane koja će spremati navedene informacije u bazu podataka MongoDB.

Nakon toga razvit će se logika koja omogućava stvaranje novih SQL upita kroz korisničko sučelje koje sadrži korake i funkcionalnosti koje u suradnji s poslužiteljskom stranom formiraju SQL upite. Jednostavan SQL upit obično se sastoji od tri glavna dijela: SELECT, FROM i WHERE. Sukladno dijelovima SQL upita razvit će se dijelovi okoline koji služe za stvaranje pojedinog dijela SQL upita. Cjelokupna okolina bit će podijeljena na dva dijela, a to su poslužiteljski i klijentski dio.

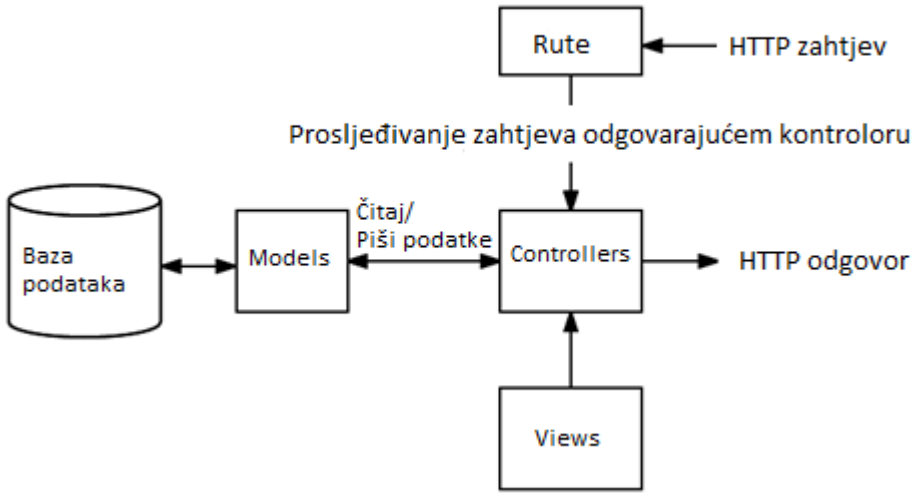
3.2. Opis modela poslužiteljske strane

Kako bi se osigurala integracija okoline sa Salesforce Marketing Cloudom te ostvarila komunikacija s klijentskom stranom na poslužiteljskoj strani primijenit će se tzv. Model–view–controller (MVC) arhitektura (Slika 3.1) pomoću koje će se stvoriti rute odnosno API-ji za komunikaciju s klijentskom stranom odnosno Salesforce Marketing Cloudom. MVC arhitekturu čine tri sloja: Models, Controllers i View. Sloj Models sadržavat će ključne objekte za ispravan rad okoline. Prilikom razvijanja logike okoline potrebno je odrediti što će biti potrebno spremiti u bazu podataka. Najprije je potrebno spremiti informacije o organizaciji kojoj korisnik pripada. Stoga će se definirati model Organizations koji će sadržavati potrebne informacije o organizaciji. Nakon toga potrebno je spremiti informacije o korisniku okoline. Stoga će se definirati model Users koji će sadržavati informacije o korisniku. Na kraju će se definirati model Selections u kojem će biti spremljene informacije za praktično korištenje okoline. Nakon što se stvore potrebni modeli okoline sljedeći korak je razvoj sloja Controllers koja će poslužiti kao posrednički sloj između klijentske strane, Salesforce Marketing Clouda i sloja Models. U sloju Controllers razvit će se funkcije bitne za ostvarivanje interakcije s bazom podataka te funkcije koje će se implementirat za korištenje resursa kojih pruža Salesforce Marketing Cloud. Također, u sloju Controllers razvit će se i logika stvaranja SQL upita čiji dijelovi će biti proslijeđeni od klijentske strane. Na kraju je potrebno razviti rute koje će služiti za komunikaciju s klijentskom stranom odnosno slojem View. Cjelokupni prikaz MVC arhitekture nalazi se na slici 3.1.



Sl. 3.1. Prikaz MVC arhitekture

Prema slici 3.1 vidljiva je interakcija između pojedinih slojeva MVC arhitekture. Sloj View surađuje sa slojem Model i Controller na način da šalje zahtjev za dohvaćanje podataka prema sloju Controller te prikazuje određene informacije koje sadrži sloj Model. Sloj Controller obrađuje zahtjeve zatražene od strane sloja View te nakon toga vraća dobivene rezultate nazad sloju View. Također, sloj Controller rukuje slojem Model tako što zahtjeva određene podatke koje mu sloj Model pruža na korištenje. Ovakav oblik arhitekture omogućuje lakše razvijanje okoline jer je okolina podijeljena u slojeve te je stoga sama logika razvijanja okoline jednostavnija budući da je jasno vidljivo koji sloj služi za koju odgovornost rada okoline. Također, aplikacije razvijene u sklopu ovog modela lakše je testirati jer je moguće odvojeno testiranje pojedinog sloja aplikacije. Osim navedenih prednosti, pomoću MVC arhitekture olakšano je i procesiranje zahtjeva koje šalje korisnik budući da postoji samo jedan sloj koji rukuje takvim zahtjevima [13]. Način protoka podataka u MVC arhitekturi prikazan je na slici 3.2.



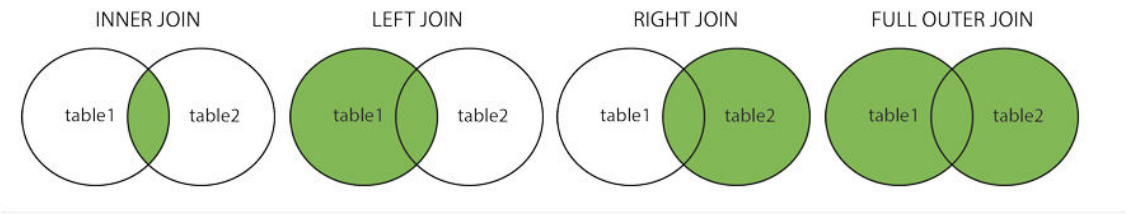
Sl. 3.2. Prikaz protoka podataka u MVC arhitekturi

Prema slici 3.2 vidljiv je način protoka podataka između pojedinih slojeva poslužiteljske strane. Prilikom zahtjeva klijenta za pristup određenoj stranici okoline, šalje se HTTP zahtjev koji preuzima sloj Rute (engl. Routes). Rute su u osnovi uzorci URL-a povezani s različitim stranicama. Svaka je ruta povezana sa slojem Controllers tj. s određenom funkcijom unutar sloja Controllers. Dakle, kada se podnese URL zahtjev, poslužiteljska strana pokušava pronaći odgovarajuću rutu te ako je ruta pronađena, poslužiteljska strana poziva funkciju sloja Controllers povezane s tom rutom. Nakon toga funkcija sloja Controllers obrađuje dobiveni zahtjev te uz korištenje sloja Models pronalazi sve potrebne podatke, organizira ih te ih šalje

sloju View koji zatim koristi te podatke za prikazivanje konačne internet stranice koja se korisniku prikazuje u pregledniku. Zahtjevi prema sloju Controllers mogu biti u svrhu dohvaćanje podataka iz baze podataka ili u svrhu manipulacije resursa Salesforce Marketing Clouda. Manipulacija resursima Salesforce Marketing Clouda postignut će se korištenjem Simple Object Access Protocola (SOAP) . SOAP se u velikoj mjeri oslanja na XML format i zajedno sa shemama definira okvir za razmjenu poruka. Kao što je uobičajeno za bilo koji XML dokument, u ovom slučaju mora postojati jedan korijenski element: omotnica (engl. envelope). Omotnica sadrži dva elementa, a to su zaglavlje i tijelo. U zaglavlju se definiraju posebni zahtjevi za poruku kao što je npr. autentifikacija dok se u tijelu definiraju sadržaji zahtjeva. Nakon što sloj Controller obavi svoj dio posla, putem API-ja definiranih u sloju Rute, prosljeđuje klijentskoj strani tražene podatke.

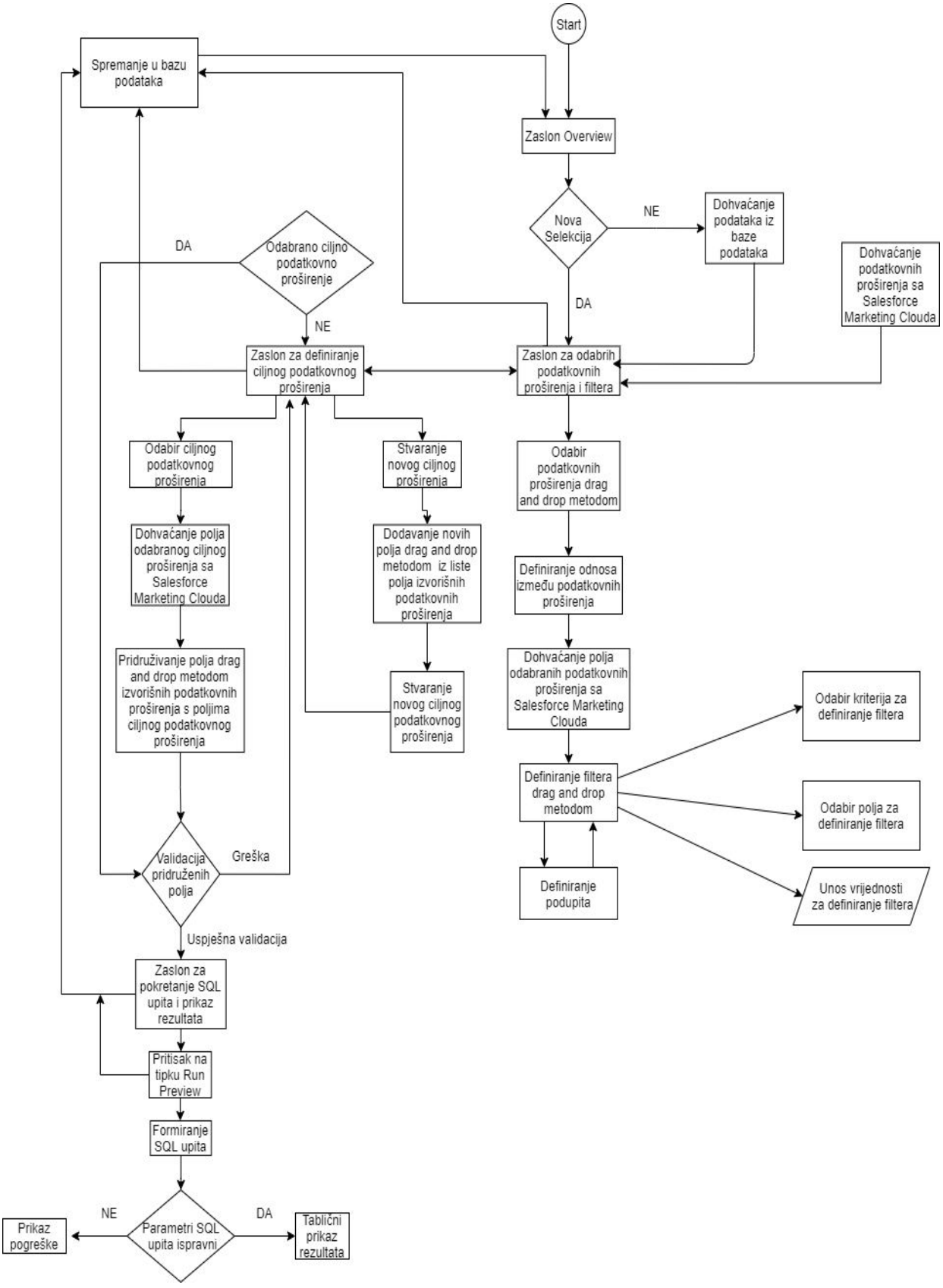
3.3. Opis modela klijentske strane

U svrhu definiranja potrebnih dijelova za stvaranje cjelokupnog SQL upita kreirat će se korisničko sučelje u kojem će se pomoću metode „*drag and drop*“ kroz tri koraka, uz suradnju s poslužiteljskom stranom, kreirati konačni SQL upit. U prvom koraku odabrat će se podatkovno proširenje na kojem se želi izvršiti upit te će se definirati filteri pomoću kojih će se vrijednosti odabranog podatkovnog proširenja ograničavati. Poslužiteljskoj strani poslat će se zahtjev za dohvat dostupnih podatkovnih proširenja koja će se na klijentskoj strani prikazati u odjeljku dostupnih podatkovnih proširenja. Podatkovna proširenja moći će se pretraživati te pomoću metode „*drag and drop*“ premjestiti u područje odabranog podatkovnog proširenja nakon čega će se prikazati polja odabranog podatkovnog proširenja. Podatkovna proširenja moći će se i povezivati tako što će korisnik metodom „*drag and drop*“ odabrati drugo podatkovno proširenje i ispusti ga na postojeće nakon čega će se pojaviti modal u kojemu je potrebno definirati polja koja ih povezuju te odnos između podatkovnih proširenja. Odnosi između podatkovnih proširenja mogu biti sljedeća: Unutarnje spajanje (engl. inner join), Lijevo spajanje (engl. left join, Desno spajanje (engl. right join) i Potpuno spajanje (engl. full outer join) (Slika 3.3) [14]



Sl. 3.3. Vrste SQL spajanja

Odabrana izvorišna podatkovna proširenja te odnos između njih spremit će na klijentskoj strani te će prilikom kreiranja konačnog SQL upita biti proslijeđeni poslužiteljskoj strani na kojoj će se kreirati klauzula FROM SQL upita. Drugim riječima, odabir izvorišnih podatkovnih proširenja i definiranja odnosa između njih odgovaraju klauzuli FROM SQL upita. Sljedeći će korak bit stvaranje klauzule WHERE SQL upita. Korisniku će biti omogućeno definiranje filtera pomoću metode „*drag and drop*“ nad poljima podatkovnog proširenja odabranog u prethodnom koraku. Zavisno o vrsti polja, korisniku će biti dostupni određeni kriteriji. Vrsta polja koja će biti dostupna za korištenje su: text, number, date, email, boolean, decimal, local i phone. Za „*date*“ vrstu polja prilikom stvaranja filtera moći će se odabrati između dva načina kriterija: calendar i relative. Način kriterija Calendar omogućit će odabir željenog datuma pomoću kalendara dok će način kriterija relative omogućiti definiranje godine, mjeseci, dana, sati prije ili poslije trenutnog datuma pokretanja SQL upita. Osim navedenih kriterija, okolina će pružiti mogućnost stvaranja pod-upita koji će sadržavati vlastite filtre uz korištenje formula kao što su: count, average, sum, minimum i maximum. Odnos između više filtera definirat će se pomoću operatora AND i OR te će također biti omogućeno mijenjanje položaja filtera, brisanje i uređivanje filtera. U sljedećem će se koraku definirati podatkovno proširenje u koje će biti spremljeni podaci nastali SQL upitom. Podatkovno proširenje odabrat će se iz padajućeg izbornika u kojem će se nalaziti sva dostupna proširenja. Nakon što je željeno proširenje odabrano, bit će potrebno određenim poljima odabranog proširenja pridružiti polja iz izvorišnog proširenja koja će se koristiti za stvaranje klauzule SELECT SQL upita. U okolini će biti moguće stvaranje novog ciljnog podatkovnog proširenja ukoliko ga korisnik nije prethodno stvorio. Također će biti omogućeno i uređivanje ciljnog proširenja odnosno mijenjanje imena i dužine polja te dodavanje novih polja ciljnom podatkovnom proširenju. Nakon što su se odabrala podatkovna proširenja iz kojih će se koristiti podaci i u koje će se ti podaci spremati, definirani filteri te pridružena polja za stvaranje SQL upita pritiskom na tipku za pokretanje pregleda dobit će se rezultati nastali izvršavanjem SQL upita. Nastali rezultati prikazat će se korisniku te ukoliko je korisnik zadovoljan dobivenim rezultatima pritiskom na tipku za pokretanje upita rezultati će biti upisani u ciljno podatkovno proširenje u Salesforce Marketing Cloudu. Cjelokupni model klijentske strane prikazan je na slici 3.4.



Sl. 3.4. Prikaz modela klijentske strane

4. PROGRAMSKO RJEŠENJE OKOLINE ZA STVARANJE SLOŽENOG SUSTAVA SQL UPITA

U ovom poglavlju prikazat će se cjelokupno programsko rješenje za izradu okoline za stvaranje složenog sustava SQL upita. Također su opisani korišteni alati i tehnologije potrebne za razvoj okoline, te je prikazano programsko rješenje po komponentama u kome su dani bitni kodovi za ostvarivanje razvoja okoline.

4.1. Razvoj i način rada poslužiteljske strane

Osnovni cilj poslužiteljske strane je povezivanje okoline sa Salesforce Marketing Cloudom kako bi se omogućilo korištenje API-ja u cilju potpune integracije sa Salesforce Marketing Cloudom. Nadalje, na poslužiteljskoj strani stvorena je arhitektura za rad i razvoj složenih SQL upita koji će se koristiti za rješavanje problematike segmentiranja korisnika u svrhu razvoja marketinških promidžbi. Također, omogućeno je i povezivanje i rad s bazom podataka te komunikacija s klijentskom stranom.

4.1.1. Programski alati i tehnologije na poslužiteljskoj strani

Poslužiteljska strana razvijena je u okruženju Visual Studio Code koji služi za uređivanje koda. Tehnologija korištena za razvoj poslužiteljske strane je Node.js pomoću kojeg je ostvareno stvaranje SQL upita, rad s bazom podataka MongoDB te implementacije potrebnih API- ja za komunikaciju s SFMC- om i klijentskom stranom. Kako bi se omogućilo lakše razvijanje okoline na lokalnom računalu korištena je višeplatformska aplikacija Ngrok.

4.1.1.1. Visual Studio Code

Visual Studio Code uređivač je otvorenog koda koji se pokreće na radnoj površini i dostupan je za Windows, MacOS i Linux. Dolazi s ugrađenom podrškom za JavaScript, TypeScript i Node.js i ima bogat ekosistem proširenja za druge jezike (poput C ++, C #, Java, Python, PHP, Go) i vremena izvođenja (kao što su .NET i Unity) [15].

4.1.1.2. Node.js

Node.js je platforma na strani poslužitelja izgrađena na JavaScript-u Google Chrome (V8 Engine). Node.js razvio je Ryan Dahl 2009. godine, a njegova najnovija inačica je v0.10.36. Aplikacije Node.js –a napisane su u JavaScript-u i mogu se izvoditi na operacijskim sustavima OS X, Microsoft Windows i Linux. Node.js također nudi bogatu biblioteku različitih JavaScript modula što u velikoj mjeri pojednostavljuje razvoj web aplikacija koje

koriste Node.js [16]. Korisnici Node.js-a ne moraju brinuti o problemu postupka „*deadlocks*“. Gotovo nijedna funkcija u Node.js izravno ne vrši I / O, tako da se postupak nikad ne blokira. Budući da se ništa ne blokira, skalabilne sustave je vrlo razumno razvijati u Node.js-u [17].

4.1.1.3. MongoDB

MongoDB je baza podataka otvorenog koda i vodeća baza podataka NoSQL tipa. Napisan je u C++ programskom jeziku.

NoSQL baze podataka nisu tablične i pohranjuju podatke drugačije od relacijskih tablica. Dolaze u različitim vrstama na temelju njihovog modela podataka. Glavne su vrste dokument, ključ-vrijednost, široki stupac i graf. Oni pružaju fleksibilne sheme i lako rukuju s velikim količinama podataka i velikim opterećenjima korisnika [18].

4.1.1.4. Ngrok

Ukoliko se želi lokalno testirati aplikacija, razvojno računalo ne može primati nikakve obavijesti zato što se ne testira na javnom internetu. Jedan od rješenja je korištenje besplatnog alata Ngrok. Ngrok stvara siguran tunel koji povezuje internet s lokalnom aplikacijom [19]. Alat čini da se internet poslužitelj na lokalnom računalu nalazi na poddomeni ngrok.com, što znači da nije potreban javni IP ili naziv domene na lokalnom računalu. Slična funkcionalnost može se postići s Reverse SSH Tunneling-om, ali to zahtijeva više podešavanja, kao i hosting vlastitog udaljenog poslužitelja. Ngrok može zaobići ograničenja NAT mapiranja i zaštitnog zida tako što će stvoriti dugovječni TCP tunel iz nasumično generirane poddomene na ngrok.com (npr. 4gf735ks.ngrok.com) na lokalnom računalu. Nakon što se odredi koji port sluša internet poslužitelj, Ngrok klijentski program pokreće sigurnu vezu s Ngrok poslužiteljem i tada svatko može postaviti zahtjeve na lokalni poslužitelj s jedinstvenom adresom Ngrok tunela [20]. Prikaz pokrenutog Ngrok tunela vidljivo je na slici 4.1.

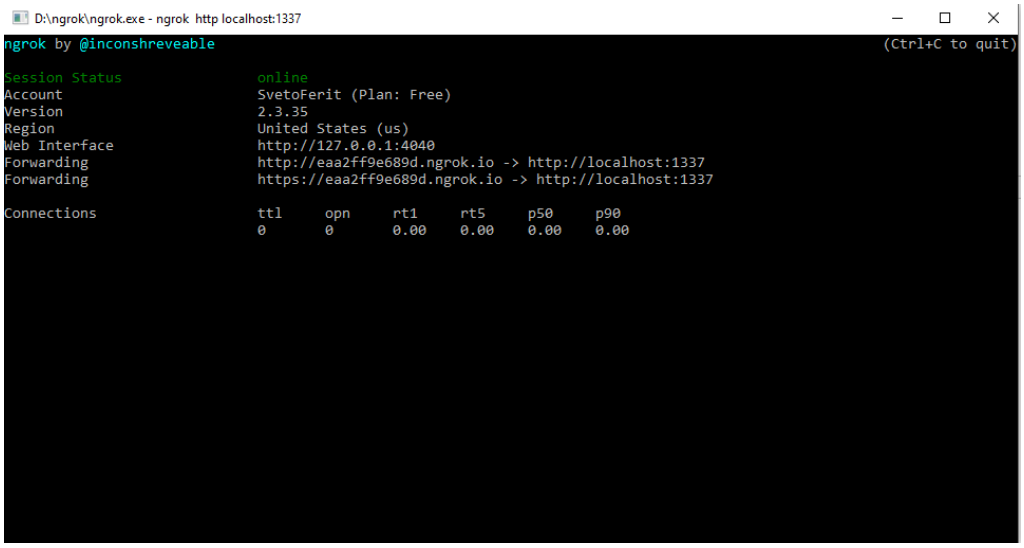
Sl. 4.1. Prikaz pokrenutog Ngrok tunela

4.1.1.5. Structured Query Language (SQL)

SQL je strukturni jezik za izradu, traženje, ažuriranje i brisanje podataka iz relacijskih baza podataka. SQL naredbe mogu se podijeliti u dva glavna pod-jezika: DDL i DML. Jezik definicije podataka (engl. DDL) sadrži naredbe koje se koriste za stvaranje i destrukciju baza podataka i objekata baze podataka. Neke od naredbi DDL-a su: CREATE, USE, ALTER, DROP. Nakon što je struktura baze podataka definirana s DDL-om, administratori baze podataka i korisnici mogu koristiti jezik upravljanja podacima (engl. DML) za umetanje, preuzimanje i izmjenu podataka sadržanih u njoj. Neke od naredbi DML-a su: INSERT, SELECT, UPDATE, DELETE [21].

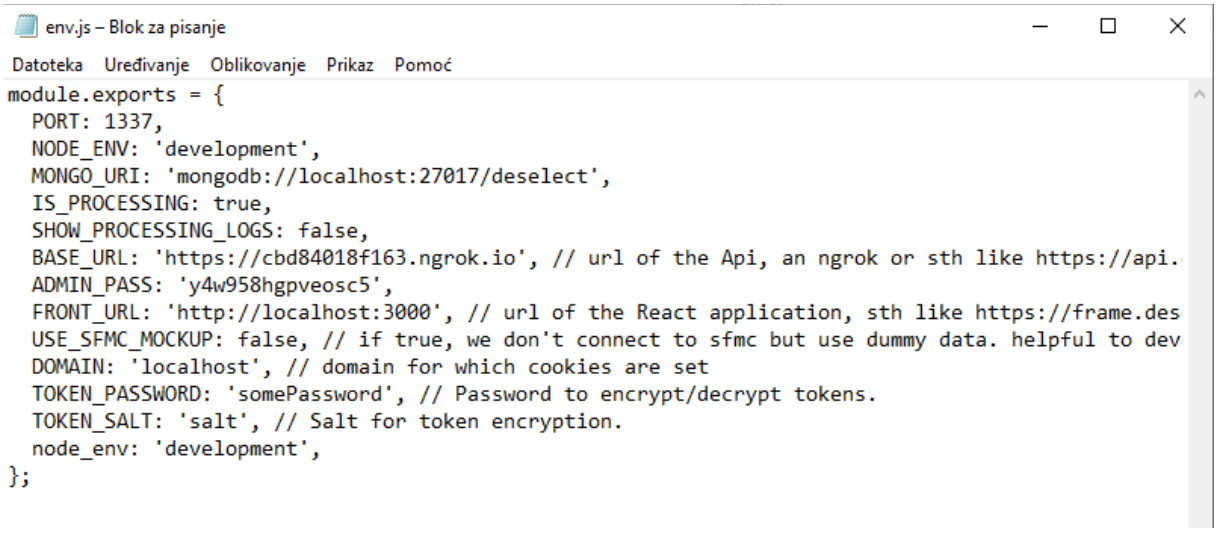
4.1.2. Kongifuriranje poslužiteljske strane

Kako bi bilo moguće razvijanje okoline na lokalnom računalu potrebno je najprije instalirati i pokrenuti alat Ngrok pomoću naredbe: ./ngrok http localhost:1337 ukoliko se razvija na MacOS-u odnosno pomoću naredbe: ngrok http localhost:1337 ukoliko se razvija na Windows OS-u. Nakon toga će se pokrenuti Ngrok tunel kao što je prikazano na slici 4.2.



Sl. 4.2. Prikaz pokrenutog Ngrok tunela

Sljedeći je korak pokretanje baze podataka MongoDB te zatim pokretanje poslužitelja. No, prije pokretanja samog poslužitelja potrebno je stvoriti i konfigurirati dokument okoline prema postavkama na slici 4.3.



Sl. 4.3. Prikaz postavki datoteke env.js

Pristup (engl. Port) je potrebno postaviti na 1337 gdje će biti pokrenut poslužitelj. NODE_ENV: development – ukazuje na to da će okolina biti pokrenuta u razvojnom načinu.

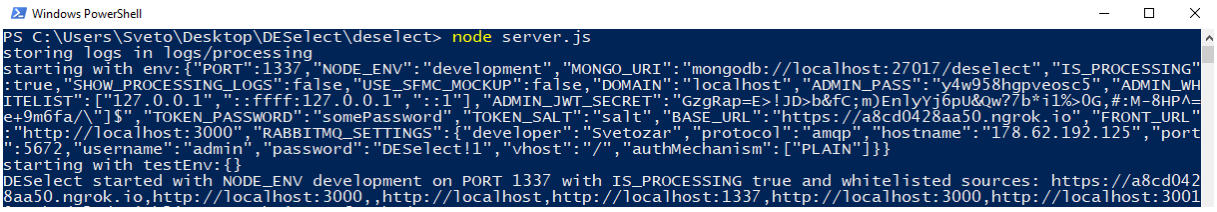
MONGO_URI: mongodb://localhost:27017/deselect – URL potreban za povezivanje s lokalnom bazom podataka.

BASE_URL - potrebno postaviti na URL generiran pokretanjem alata Ngrok.

ADMIN_PASS – potreban za stvaranje nove organizacije unutar lokalne baze podataka.

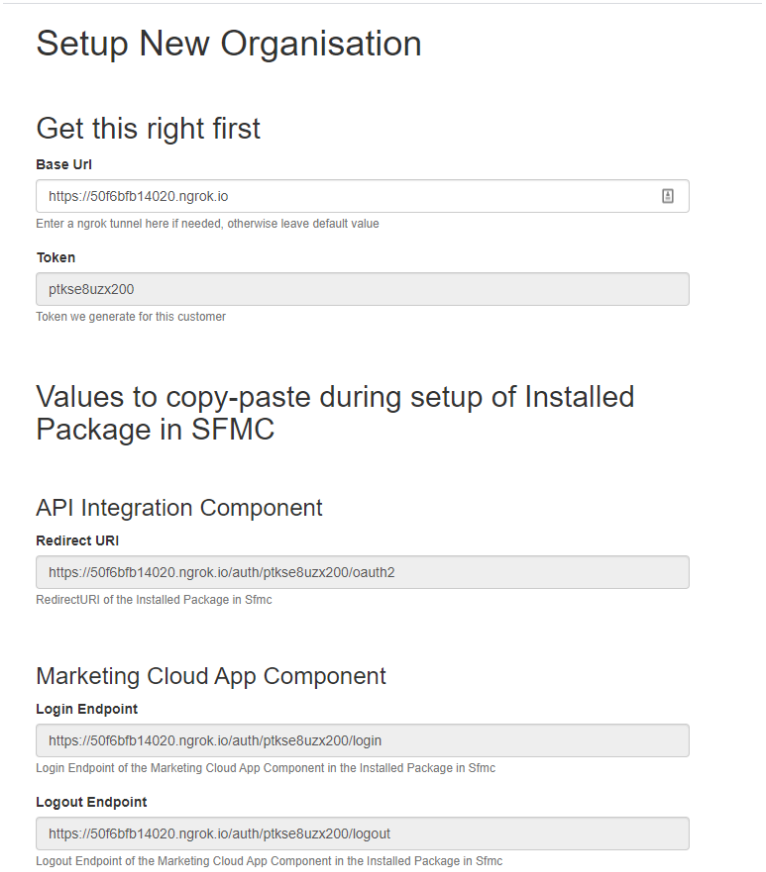
FRONT_URL – URL na kojem će klijentska strana biti pokrenuta.

Nakon što su se definirali potrebne parametre potrebno je pokrenuti poslužitelj naredbom node sever.js (Slika 4.4).



Sl. 4.4. Prikaz pokrenutog poslužitelja

Nakon što je poslužitelj pokrenut potrebno je stvoriti novu organizaciju kako bi bilo omogućeno korištenje okoline. Pomoću čarobnjaka za instalaciju potrebno je stvoriti novu organizaciju kao što je prikazano na slici 4.5.

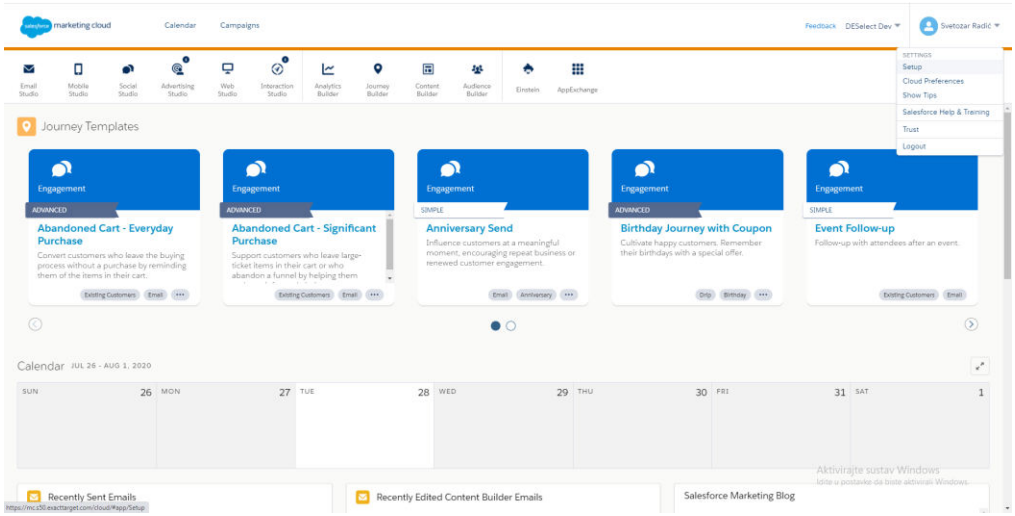


Sl. 4.5. Prikaz kreiranja nove organizacije

Prema slici 4.5 vidljivo je stvaranje novih poveznica potrebne za stvaranje novog paketa u Salesforce Marketing Cloudu. Poveznica generirana pomoću Ngrok tunela koristi se kao osnovni URL (engl. *Base URL*) kako bi se okolina koja je pokrenuta na lokalnom računalu

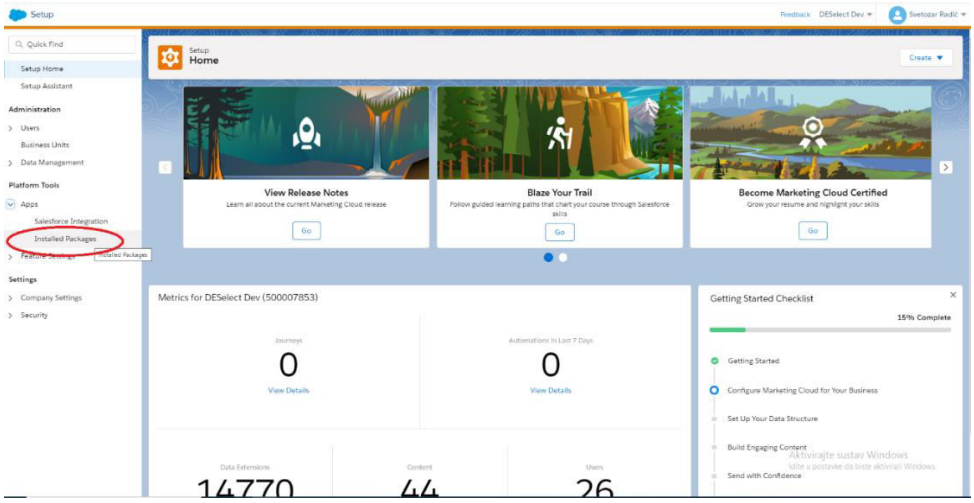
bila integrirana sa Salesforce Marketing Cloudom. Nakon što se unese osnovni URL generira se token pomoću kojeg se stvore *Login Endpoint* i *Logout Endpoint* potrebni za stvaranje novog paketa u Salesforce Marketing Cloudu. Prije završetka stvaranja nove organizacije potrebno je prijaviti se u Salesforce Marketing Cloud te učiniti korake prikazane slikom :

- Kliknuti svoje ime u gornjem desnom kutu → Setup (Slika 4.6)



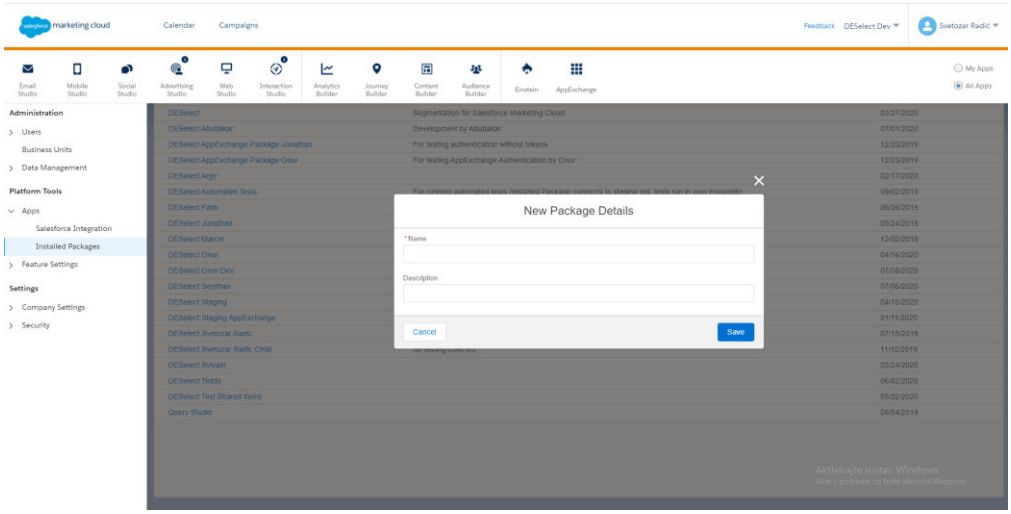
Sl. 4.6. Prikaz pristupa *Setup* postavkama korisničkog računa

- Nakon toga → Apps → Installed Packages (Slika 4.7)



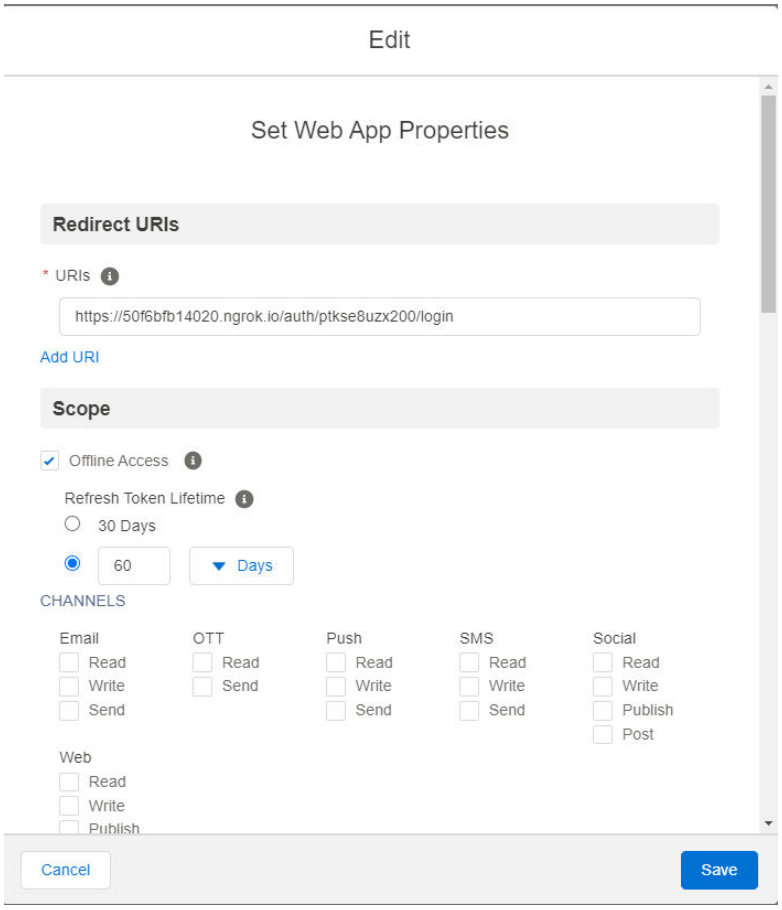
Sl. 4.7. Prikaz pristupa Installed Packages postavkama

- Pritiskom na „Installed Packages“ otvorit će se novi prozor u kojem je potrebno stvoriti novi paket pritiskom na tipku New nakon čega će se otvoriti prozor u kojem je potrebno definirati ime i opis novog paketa (Slika 4.8)

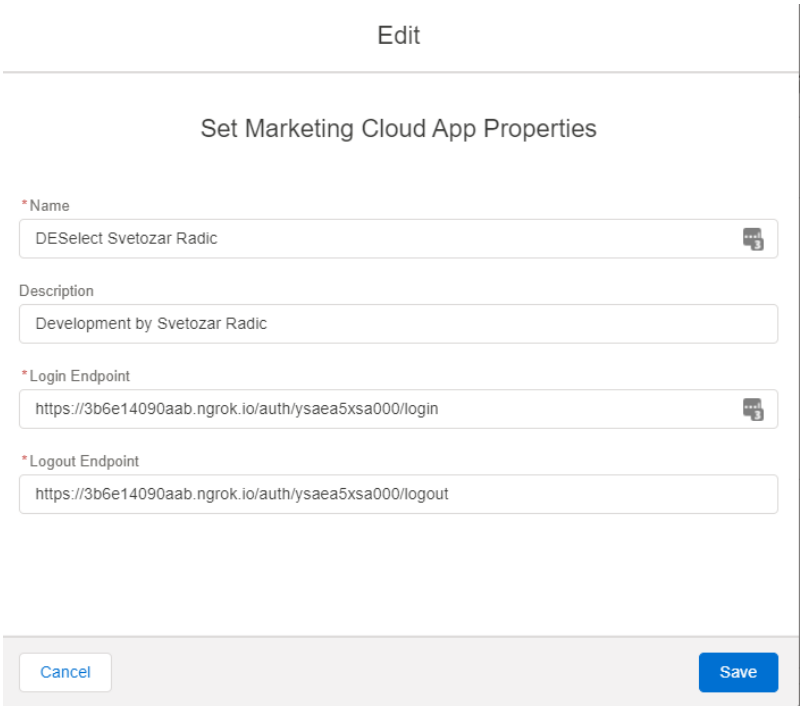


Sl. 4.8. Prikaz unosa informacija o novom paketu

- Nakon što je stvoren novi paket potrebno je dodati sljedeće komponente: Komponentu API Integration i komponentu Marketing Cloud App te pomoću poveznica generiranih pomoću čarobnjaka za instalaciju konfigurirati navedene komponente prema slikama 4.9 i 4.10.



Sl. 4.9. Prikaz konfiguracije komponente API Integration

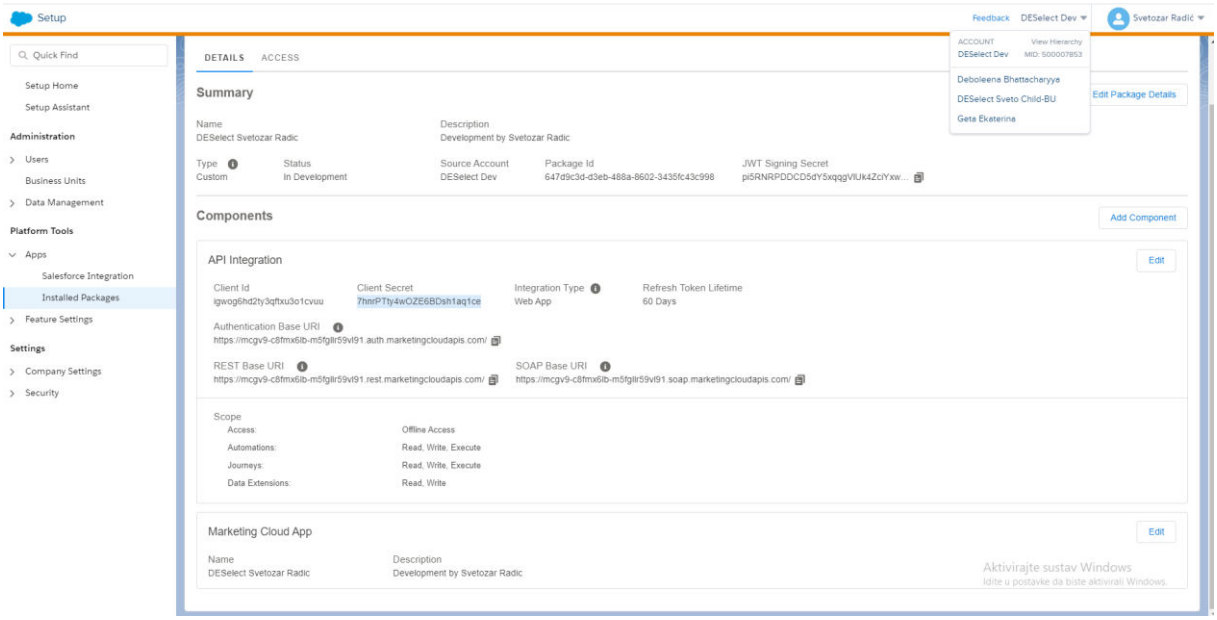


Sl. 4.10. Prikaz konfiguracije komponente Marketing Cloud App

Prema slikama 4.9 i 4.10 vidljivo je da su poveznice *Redirect URI* te poveznice *Login Endpoint* i *Logout Endpoint* prethodno generirane prilikom stvaranja nove organizacije.

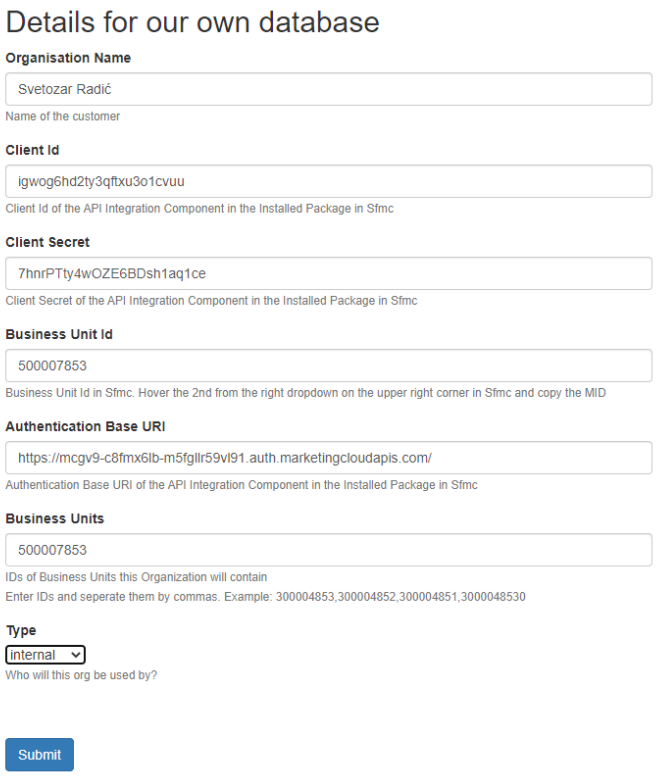
4.1.3. Povezivanje s bazom podataka MongoDB

Nakon stvaranja novog paketa u Salesforce Marketing Cloudu potrebno je popuniti podatke u čarobnjaku za instalaciju kako bi se uspješno stvorila nova organizacija unutar lokalne baze podataka kao što je prikazano na slikama 4.9 i 4.10.



Sl. 4.11. Prikaz instaliranog paketa u Salesforce Marketing Cloud- u

Prema slici 4.11 prikazani su parametri novokreiranog paketa kao što su Client Id, Client Secret, Authentication Base URI te MID koji su potrebni pri procesu kreiranja nove organizacije prikazanog na slici 4.12.



Sl. 4.12. Prikaz procesa kreiranja nove organizacije

Pritiskom na tipku Submit, prikazanog na slici 4.12, stvara se nova organizacija u lokalnoj bazi podataka. Kreirana organizacija unutar baze podataka MongoDB prikazana je na slici 4.13.

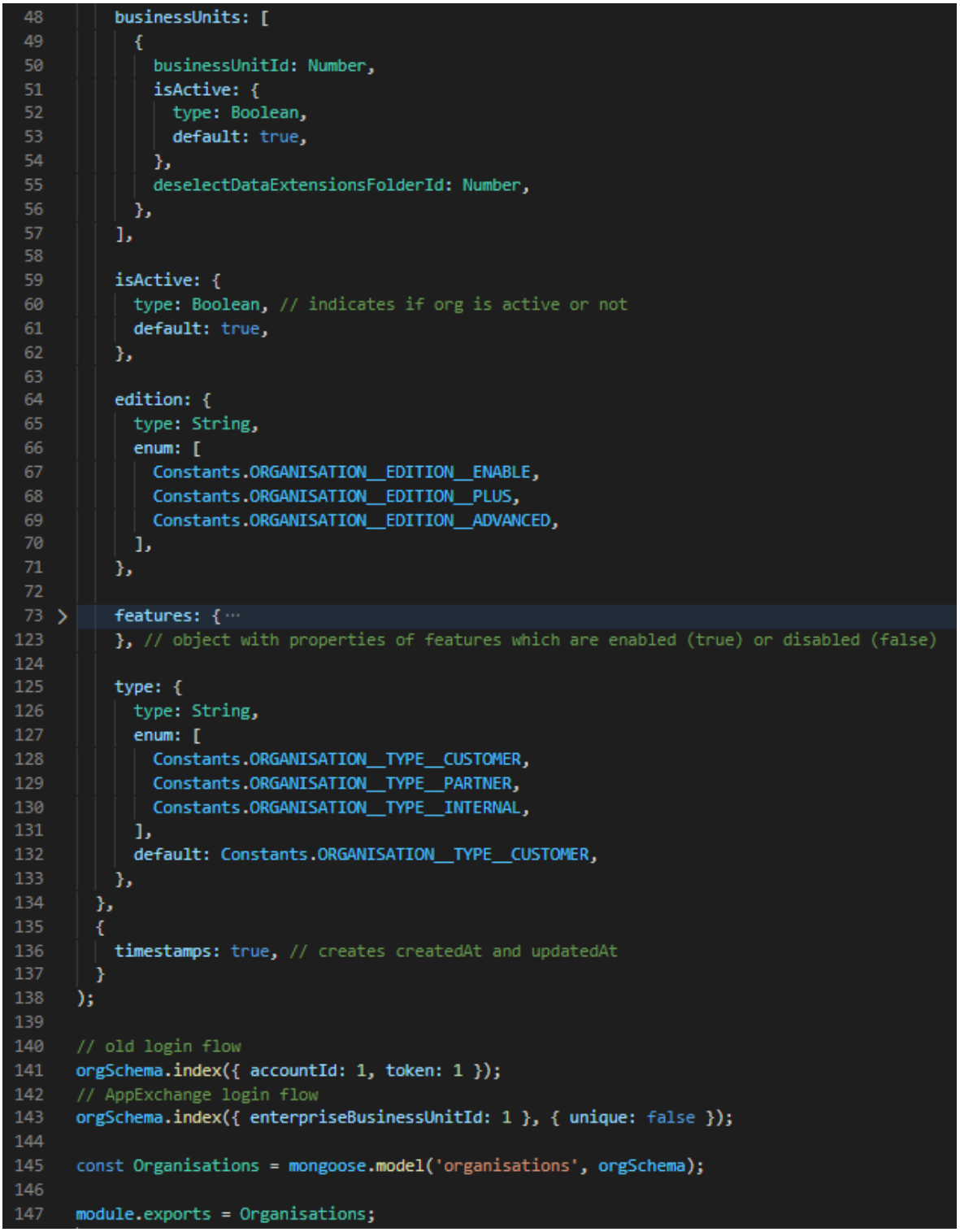

```
_id: ObjectId("5efb276cbbee312c84cc6055")
updatedAt: 2020-06-30T14:16:43.024+00:00
createdAt: 2020-06-30T11:52:12.374+00:00
name: "DESelect Svetozar Radić"
token: "ysaea5xsa000"
clientId: "igwog6hd2ty3qftxu3o1cvuu"
clientSecret: "7hnrPTty4wOZE6BDsh1aq1ce"
accountId: "500007853"
authenticationBaseUrl: "https://mcgv9-c8fmx6lb-m5fgllr59vl91.auth.marketingcloudapis.com/"
marketingCloudSubdomain: "mcgv9-c8fmx6lb-m5fgllr59vl91"
type: "internal"
> features: Object
isActive: true
> businessUnits: Array
includeOrExcludeTargetDEFolders: "exclude"
includeOrExcludeAvailableDEFol…: "exclude"
> targetDEFoldersToFilter: Array
> availableDEFoldersToFilter: Array
__v: 0
enterpriseBusinessUnitId: 500007853
marketingCloudStackNumber: "S50"
```


Sl. 4.13. Prikaz novokreirane organizacije

Kako bi se kreirala nova organizacija unutar baze podataka potrebno je najprije definirati model organizacije odnosno shemu koja će biti spremljena u bazu podataka. Model organizacije prikazan je na slikama 4.14 i 4.15.

```
const mongoose = require('mongoose');

const Constants = require('@controllers/constants');

const orgSchema = new mongoose.Schema(
  {
    name: String,
    token: {
      type: String,
      index: { unique: true },
    },
    accountId: {
      type: String,
    },
    clientId: String, // Never expose this in a frontend call!
    clientSecret: String, // Never expose this in a frontend call!
    authenticationBaseUrl: String, // eg: https://mc6qx5nmngnzzxdd2fh3h3m0zb98.auth.marketingcloudapis.com/
    marketingCloudSubdomain: String, // eg mc6qx5nmngnzzxdd2fh3h3m0zb98
    marketingCloudStackNumber: String, // eg s50, s8
    // ObjectId of the folder for DEs we create in sfmc
    deselectDataExtensionsFolderId: String,
    // List of DE Folder IDs to be excluded in Available Data Extensions
    availableDEFoldersToFilter: {
      type: Array,
      default: [],
    },
    // List of DE Folder IDs to be excluded in Target Data Extensions
    targetDEFoldersToFilter: {
      type: Array,
      default: [],
    },
    // Determine if Available DE Folders List will be included or excluded
    includeOrExcludeAvailableDEFolders: {
      type: String,
      default: Constants.ORGANISATION__INCLUDE_OR_EXCLUDE_TARGETDE_FOLDERS__EXCLUDE,
    },
    // Determine if Target DE Folders List will be included or excluded
    includeOrExcludeTargetDEFolders: {
      type: String,
      default: Constants.ORGANISATION__INCLUDE_OR_EXCLUDE_TARGETDE_FOLDERS__EXCLUDE,
    },
    // ObjectId of the folder for QueryActivities we create in sfmc
    deselectQueryActivitiesFolderId: String,
    // Id of the highest parent business unit, would normally be unique (but not enforced because the old login method
      allows to have multiple instances of DESelect with different tokens installed in the same org)
    enterpriseBusinessUnitId: {
      type: Number,
    },
    businessUnits: [
```

Sl. 4.14. Prikaz modela organizacije (A)



Sl. 4.15. Prikaz modela organizacije (B)

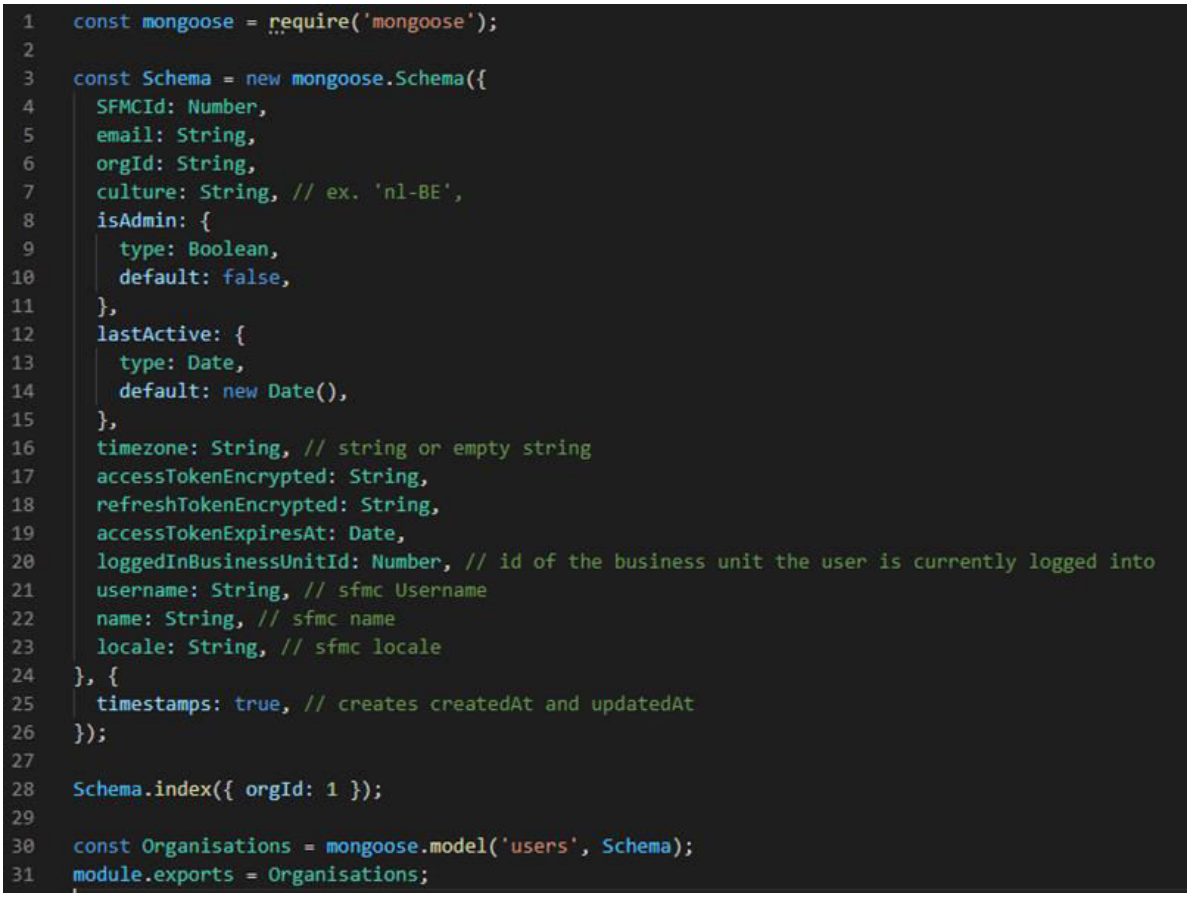
Na slikama 4.14 i 4.15 prikazana je mongoose shema organizacije koja će biti spremljena u bazu podataka. Također, za svaki atribut sheme (npr. accountId, clientId, itd,…) definiran je tip podatka (npr. String, Boolean, itd…) te komentar kao bitna informacija o pojedinom atributu. Nakon definiranja modela kreira se funkcija pomoću koje se nova organizacija

sprema u bazu podataka. Spremanje nove organizacije unutar baze podataka MongoDB prikazana je kodom na slici 4.16.

```
/**
 * Creates a new Organisation.
 * @param {String} name - Name of the Organisation.
 * @param {String} token - Access Token.
 * @param {String} clientId - Client ID.
 * @param {String} clientSecret - Client Secret.
 * @param {String} accountId - Account ID.
 * @param {String} authenticationBaseUrl - Authentication URL.
 * @param {String} marketingCloudSubdomain - MarketingCloudSubdomain.
 * @param {String[]} businessUnitIds - Array of Business Unit IDs defined for the Org.
 * @param {String} type - Type of the Organisation.
 * @returns {Promise<Object>} - Created Organisation.
 */
const create = async (name, token, clientId, clientSecret, accountId, authenticationBaseUrl, marketingCloudSubdomain,
  businessUnitIds, type) => {
  const businessUnits = [];
  businessUnitIds.forEach((businessUnitId) => {
    businessUnits.push({
      businessUnitId: parseInt(businessUnitId),
      isActive: true,
    });
  });

  try {
    const org = new Orgs({
      name,
      token,
      clientId,
      clientSecret,
      accountId,
      authenticationBaseUrl,
      marketingCloudSubdomain,
      businessUnits,
      type,
    });
    const newOrg = await org.save();

    return new Promise((resolve) => {
      resolve(newOrg);
    });
  } catch (error) {
    return Util.handleError(error);
  }
};
```

Sl. 4.16. Prikaz funkcije za kreiranje organizacije unutar baze podataka MongoDB

Na slici 4.16 prikazana je funkcija create() za kreiranje nove organizacije. Funkcija prima parametre kao što su name, token, cliendId, itd. Na početku funkcije se pomoću petlje spremaju identifikatori svake poslovne jedinice (engl. business unit) nakon čega se deklarira nova organizacija pomoću naredbe new Orgs() s proslijeđenim parametrima. Funkcijom save() sprema se nova organizacija u bazu podataka MongoDB. Ukoliko je organizacija uspješno spremljena funkcija create() vraća model organizacije, u suprotnom funkcija vraća grešku. Nakon što je organizacija stvorena u bazi podataka, potrebno je otvoriti samu okolinu unutar App Excange-a u Salesforce Marketing Cloudu kako bi se generirao novi korisnik unutar baze podataka. Prvo je potrebno stvoriti model korisnika odnosno shemu koja će biti spremljena u bazu podataka. Model korisnika s potrebnim svojstvima nalazi se na slici 4.17.



```
const mongoose = require('mongoose');

const Schema = new mongoose.Schema({
  SFMCId: Number,
  email: String,
  orgId: String,
  culture: String, // ex. 'nl-BE',
  isAdmin: {
    type: Boolean,
    default: false,
  },
  lastActive: {
    type: Date,
    default: new Date(),
  },
  timezone: String, // string or empty string
  accessTokenEncrypted: String,
  refreshTokenEncrypted: String,
  accessTokenExpiresAt: Date,
  loggedInBusinessUnitId: Number, // id of the business unit the user is currently logged into
  username: String, // sfmc Username
  name: String, // sfmc name
  locale: String, // sfmc locale
}, {
  timestamps: true, // creates createdAt and updatedAt
});

Schema.index({ orgId: 1 });

const Organisations = mongoose.model('users', Schema);
module.exports = Organisations;
```

Sl. 4.17. Prikaz modela korisnika

Prema slici 4.17 vidljivo je kako se pomoću funkcije mongoose.Shema() definira model korisnika s bitnim parametrima za ispravan rad okoline. Naredbom mongoose.model() stvara se novi model korisnika koji će kasnije služiti za interakciju s okolinom. Nakon toga se implementira standard OAuth 2.0 pomoću kojeg se generira token pristupa pomoću kojeg će se korisnik ovjeriti te će mu biti omogućeno korištenje okoline. Nakon što je korisnik ovjeren dobiveni podaci o korisniku spremaju se u bazu podataka pomoću funkcije prikazane na slici 4.18. Prema slici 4.18, vidljivo je kako funkcija za stvaranje novog korisnika createNewUserIfUserDoesntExist() najprije poziva funkciju findOne() nad modelom korisnika te ukoliko je korisnika pronađen funkcija jednostavno vraća pronađeni model korisnika. Ukoliko korisnik nije pronađen, što znači da se radi o novom korisniku, stvara se novi model korisnika pomoću funkcije new User() te se naredbom save() novonastali model korisnika sprema u bazu podataka. Ukoliko nije došlo do nikakve greške prilikom spremanja korisnika u bazu podataka funkcija createNewUserIfUserDoesntExist() vraća model korisnika dok ukoliko je došlo do pogreške funkcija createNewUserIfUserDoesntExist() vraća pogrešku.

```
206  /**
207   * Creates a new User with given orgId and username, if none exists already. If it does, returns the existing one.
208   * @param {String} orgId - ID of the Organisation.
209   * @param {String} username - Username.
210   * @returns {Promise<Object>} - Created or Found User.
211   */
212  const createNewUserIfUserDoesntExist = async (orgId, username, email, name) => {
213    try {
214      const user = await Users.findOne({ orgId, username }).exec();
215      console.log(`user after first find in createNewUserIfUserDoesntExist:${JSON.stringify(user)}`);
216      if (user && user._id) {
217        // Return user if found
218        return new Promise((resolve) => {
219          resolve(user);
220        });
221      }
222
223      // Mark new user automatically as admin if the organization is a partner or internal org
224      const org = await Org.findOne({ _id: orgId });
225      let isAdmin = false;
226      if (org.type === Constants.ORGANISATION__TYPE__PARTNER || org.type === Constants.ORGANISATION__TYPE__INTERNAL) {
227        isAdmin = true;
228      }
229
230      // first time login. So create user
231      console.log('creating a new user');
232      const newUser = await new Users({ orgId, username, isAdmin }).save();
233
234      // Zap: Create User in Mailchimp
235      const body = { email, name };
236      if (env.NODE_ENV === 'production') {
237        Util.postToZapier(body, 'https://hooks.zapier.com/hooks/catch/5896539/od2zb0h/');
238      }
239
240      return new Promise((resolve) => {
241        resolve(newUser);
242      });
243    } catch (error) {
244      return Util.handleError(error);
245    }
246  };
```

Sl. 4.18. Funkcija za spremanje korisnika

Nakon što su organizacija i korisnik uspješno stvoreni i spremljeni u bazu podataka može se početi koristiti okolina.

4.1.4. Prikaz programskog rješenja poslužiteljske strane po komponentama

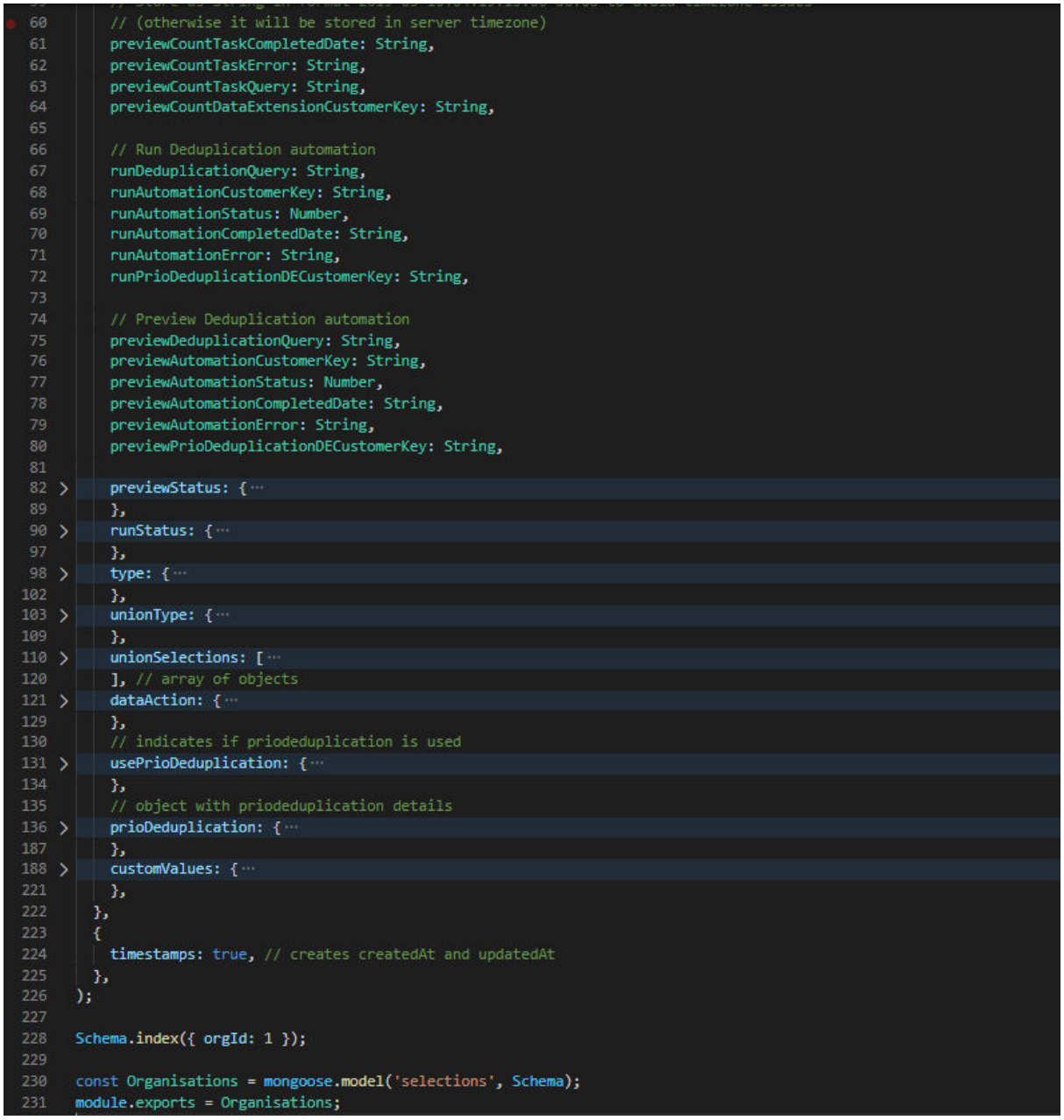
U ovom je poglavlju opisano programsko rješenje po komponentama u kome su dani bitni kodovi za ostvarivanje razvoja okoline.

4.1.4.1. Modeli aplikacije (Models)

Modeli aplikacije posjeduju ključne objekte za ispravan rad aplikacije. Kreirana su tri najbitnija modela aplikacije, a to su: model Organizations koji predstavlja model organizacije koja koristi aplikaciju (Slike 4.14 i 4.15), model Users koji predstavlja model korisnika koji koristi aplikaciju (Slika 4.17) i model Selections koji predstavlja model koji sadržava sve informacije za praktično korištenje aplikacije (Slike 4.21 i 4.22).

```
const mongoose = require('mongoose');

const Constants = require('@controllers/constants');

const Schema = new mongoose.Schema(
  {
    previewQueryActivityId: String,
    businessUnitId: Number,
    orgId: String,
    name: String,
    query: String,
    createdBy: String, // sfmc username
    createdById: String, // deselect user id
    updatedBy: String, // sfmc username
    updatedById: String, // deselect user id
    numberOfRecords: Number, // allow null and numbers
    numberOfPreviewRecords: Number,
    fields: Object,
    relations: Object,
    collections: Object,
    filters: Object, // Array of objects with operator and filters array
    targetCollectionCustomerKey: String,
    targetCollectionObjectID: String, // ObjectID of target data extension
    queryActivityId: String, // Object ID of query activity
    folderId: String,

    // Run task
    taskId: String,
    taskStatus: Number,
    // Store as String in format 2019-05-10T04:19:13.00-06:00 to avoid timezone issues
    // (otherwise it will be stored in server timezone)
    taskCompletedDate: String,
    taskError: String,
    taskQuery: String,

    // Run count task
    countTaskId: String,
    countTaskStatus: Number,
    // Store as String in format 2019-05-10T04:19:13.00-06:00 to avoid timezone issues
    // (otherwise it will be stored in server timezone)
    countTaskCompletedDate: String,
    countTaskError: String,
    countTaskQuery: String,
    countDataExtensionCustomerKey: String,

    // Preview task
    previewTaskId: String,
    previewTaskStatus: Number,
    // Store as String in format 2019-05-10T04:19:13.00-06:00 to avoid timezone issues
    // (otherwise it will be stored in server timezone)
    previewTaskCompletedDate: String,
    previewTaskError: String,
    previewTaskQuery: String,
    previewDataExtensionCustomerKey: String,

    // Preview Count task
    previewCountTaskId: String,
    previewCountTaskStatus: Number,
    // Store as String in format 2019-05-10T04:19:13.00-06:00 to avoid timezone issues
    // (otherwise it will be stored in server timezone)
```

Sl. 4.21. Prikaz modela selekcije (A)



Sl. 4.22. Prikaz modela selekcije (B)

Na slici 4.22 vidljivo je kako model Selections sadrži polja bitna za definiranje SQL upita te jednostavno korištenje istih. Neka od najbitnijih polja su: name (ime selekcije koja je kreirana), createdBy (ime korisnika koji je kreirao selekciju), updateBy (ime korisnika koji je promijenio selekciju), fields (polja podatkovnog proširenja korištena za stvaranje SQL upita), filters (filteri za stvaranje SQL upita), relations (odnosi između podatkovnih proširenja), collections (izvorišna podatkovna proširenja), tagetCollectionObjectId (jedinstveni indetifikator ciljnog podatkovnog proširenja), numberOfRecords (broj ukupnih rezultata nastalih SQL upitom), folderId (jedinstveni identifikator datoteke u kojoj je selekcija locirana) i query (krajnji SQL upit).

4.1.4.2. Upravljači (Controllers)

U datoteci Controllers smještene su funkcije koje pomažu pri integriranju okoline sa Salesforce Marketing Cloudom te omogućavaju bitne funkcionalnosti okoline kao što su datotečna organizacija, kreiranje složenih SQL upita te rukovanje s bazom podataka. Kako bi aplikacija rješavala problem pisanja SQL upita unutar Salesforce Marketing Clouda jedan od glavnih zahtjeva je i sama integracija sa Salesforce Marketing Cloudom. Integracija sa Salesforce Marketing Cloudom ostvarena je kreiranjem funkcija koje implementiraju Salesforce Marketing Cloud SOAP API-je za ostvarivanje različitih funkcionalnosti. Spomenute funkcije smještene su unutar pod – datoteke sfmc.

Funkcije koristeći SOAP API-je služe za sljedeće funkcionalnosti:

- Rukovanje SQL upitima unutar Automation Studio-a unutar Salesforce Marketing Clouda. Automation Studio je aplikacija Salesforce Marketing Clouda koja se koristi za izvršavanje marketinških aktivnosti i upravljanja podacima pomoću SQL upita. Funkcije omogućuju stvaranje, brisanje i uređivanje SQL upita. Primjer funkcije za kreiranje SQL aktivnosti prikazan je na slici 4.23. Prema slici 4.23 vidljivo je kako funkcija create() najprije pomoću for petlje prolazi kroz sve aktivnosti te ih spaja u jedan dio SOAP API-ja te nakon toga stvara tijelo SOAP API-ja [22] nakon čega se pomoću funkcije execute() stvoreni SOAP API stvara i pokreće. Nakon što je poziv završen provjerava se da li je došlo do pogreške te ukoliko je došlo do pogreške dobivena pogreška se vraća u suprotnom se vraćaju dobiveni podaci. Funkcija create() prima sljedeće parametre: authConfig (vjerodajnice za provjeru autentičnosti), businessUnitId (jedinstveni identifikator poslovne jedinice kojoj korisnik pripada), name (naziv automatizacije), description (opis automatizacije), activities (skup aktivnosti), folderId (jedinstveni identifikator datoteke u kojoj će SQL biti kreiran)

```
/**
 * Creates Automation Object.
 * @param {Object} authConfig - Authorization Credentials.
 * @param {String} businessUnitId - Business Unit ID.
 * @param {String} name - Automation name.
 * @param {String} description - Description for Automation.
 * @param {Object[]} activities - Array of activity objects.
 * @param {Number} folderId - Folder's ID to create Automation in.
 * @returns {Promise<Object>} - Created Automation Object.
 */
const create = (authConfig, businessUnitId, name, description, activities, folderId) => {
  // Create Payload for Activities.
  let automationTasks = '';
  for (let k = 0; k < activities.length; k += 1) {
    automationTasks += `<AutomationTask>
      <PartnerKey xsi:nil="true"/>
      <ObjectID xsi:nil="true"/>
      <Name>${activities[k].Name}</Name>
      <Activities>
        <Activity>
          <PartnerKey xsi:nil="true"/>
          <ObjectID>${activities[k].ObjectID}</ObjectID>
          <Name>${activities[k].Name}</Name>
          <ActivityObject xsi:type="QueryDefinition">
            <PartnerKey xsi:nil="true"/>
            <ObjectID>${activities[k].ObjectID}</ObjectID>
            <Name>${activities[k].Name}</Name>
          </ActivityObject>
        </Activity>
      </Activities>
    </AutomationTask>`;
  }

  let payload = `<soapenv:Body>
  <CreateRequest xmlns="http://exacttarget.com/wsdl/partnerAPI">
     <Options/>
     <Objects xsi:type="Automation">
        <Client>
           <ID>${businessUnitId}</ID>
        </Client>
        <Name>${name}</Name>
        <Description>${description}</Description>\n`;
  if (folderId) {
    payload += `<CategoryID>${folderId}</CategoryID>\n`;
  }
  payload += `<AutomationTasks>${automationTasks}</AutomationTasks>
        <AutomationType>scheduled</AutomationType>
     </Objects>
  </CreateRequest>
</soapenv:Body>`;

  return new Promise((resolve, reject) => {
    soap.execute(authConfig, Constants.SFMC_SOAP_ACTION__CREATE, Constants.SFMC_OBJECT__AUTOMATION, payload, (error,
      data) => {
      if (error) {
        Util.handleError(error, reject);
      } else if (data && data.Results && data.OverallStatus) {
        if (data.OverallStatus === 'Error') {
          Util.handleError(data.Results.StatusMessage, reject);
        } else {
          resolve(data.Results.Object);
        }
      } else if (data && data['soap:Fault']) {
        error = Util.getErrorMessageFromSoapFault(data['soap:Fault']);
        Util.handleError(error, reject);
      } else {
        Util.handleError('No Results From SFMC While Creating Automation', reject);
      }
    });
  });
};
```

Sl. 4.23. Prikaz funkcije za kreiranje SQL aktivnosti

- Rukovanje podatkovnim proširenjima (engl. Data Extensions) koje čine osnovu podataka za kontaktiranje unutar Salesforce Marketing Clouda. Stvaranjem i povezivanjem podatkovnih proširenja može se pristupiti informacijama o kontaktima koji se koriste za segmentiranje, filtriranje i ciljanje kontakata s relevantnim slanjima.

Ti podaci mogu se koristiti i za ciljanje kontakata na više kanala, poput e-pošte, društvenih mreža i mobilnih poruka. Također, mogu se koristiti podatkovna proširenja za spremanje podataka uvezenih iz vanjskih izvora koji se koriste kao dio aktivnosti segmentiranja. Funkcije za rukovanje podatkovnim proširenjima omogućuju dohvaćanje svih podatkovnih proširenja, njihova polja te detaljne informacije koje se nalaze u Salesforce Marketing Cloudu. Ostvarena je i mogućnost stvaranja novih podatkovnih proširenja kao i njihovo uređivanje. Primjer funkcije za stvaranje novog podatkovnog proširenja prikazan je na slici 4.24.

```
/**
 * Creates a new Data Extension.
 * @param {Object} authConfig - Authorization Credentials.
 * @param {Object} input - Properties of the Data Extension.
 * @param {Object} retentionPolicy - Retention settings of the Data Extension.
 * @param {Object} relationship - Relationship settings of the Data Extension.
 * @returns {Promise<Object>} - Created Data Extension.
 */
const create = (authConfig, input, retentionPolicy, relationship) => {
  // Create string for fields
  const fields = createFieldsString(input.fields);

  const customerKey = xml.escapeXML(input.customerKey || input.name);
  let payload = `<soapenv:Body>
                  <CreateRequest xmlns="http://exacttarget.com/wsdl/partnerAPI">
                    <Options></Options>
                      <Objects xmlns:ns1="http://exacttarget.com/wsdl/partnerAPI" xsi:type="ns1:DataExtension">
                        <CustomerKey>${customerKey}</CustomerKey>
                        <Name>${input.name}</Name>
                        <Description>${input.description}</Description>\n`;
  if (input.folderId) {
    payload += `          <CategoryID>${input.folderId}</CategoryID>\n`;
  }

  if (retentionPolicy && retentionPolicy.dataRetentionPolicy) {
    payload += createRetentionString(retentionPolicy);
  }
  if (relationship && relationship.isTestable) {
    payload += `          <IsTestable>${relationship.isTestable}</IsTestable>`;
  }
  if (relationship && relationship.isSendable && relationship.sendableDataExtensionField.name && relationship.
    sendableSubscriberField.name) {
    payload += `          <IsSendable>${relationship.isSendable}</IsSendable>
                        <SendableDataExtensionField>
                          <FieldType>${relationship.sendableDataExtensionField.type}</FieldType>
                          <Name>${relationship.sendableDataExtensionField.name}</Name>
                        </SendableDataExtensionField>
                        <SendableSubscriberField>
                          <Name>${relationship.sendableSubscriberField.name}</Name>
                        </SendableSubscriberField>`;
  }
  payload += `          <Fields>
                        ${fields}
                        </Fields>
                      </Objects>
                    </CreateRequest>
                  </soapenv:Body>`;

  return new Promise((resolve, reject) => {
    soap.execute(authConfig, Constants.SFMC_SOAP_ACTION__CREATE, Constants.SFMC_OBJECT__DATA_EXTENSION, payload,
      (error, data) => {
      if (error) {
        Util.handleError(error, reject);
      } else if (data && data.Results && data.OverallStatus) {
        if (data.OverallStatus === 'Error') {
          Util.handleError(data.Results.StatusMessage, reject);
        } else {
          data.Results.Object.Fields.Field = Array.isArray(data.Results.Object.Fields.Field) ?
            data.Results.Object.Fields.Field :
            [data.Results.Object.Fields.Field];
          resolve(data.Results.Object);
        }
      } else if (data && data['soap:Fault']) {
        error = Util.getErrorMessageFromSoapFault(data['soap:Fault']);
        Util.handleError(error, reject);
      } else {
        Util.handleError('No results from SFMC while creating DataExtension', reject);
      }
    });
  });
};
```

Sl. 4.24. Prikaz funkcije za stvaranje podatkovnog proširenja

Prema slici 4.24 vidljivo je kako funkcija create() prima sljedeće parametre: authConfig (vjerodajnice za provjeru autentičnosti), input (svojstva podatkovnog proširenja), retentionPolicy (postavke zadržavanja), relationship (postavke odnosa

podatkovnog proširenja). Funkcija create() pomoću funkcije createFieldString() kreira dio SOAP API-ja polja korištenih za stvaranje novog podatkovnog proširenja. Nakon toga se pomoću funkcije excapeXML() očiste specijalni XML znakovi kao što su &, <, >. Na kraju se u funkciji create() stvara tijelo SOAP API-ja [23] koji se izvršava funkcijom execute() te se nakon izvršavanja provjerava ukoliko je došlo do pogreške ili su podaci ispravno dohvaćeni.

- Rukovanje datotekama sadržanih u Salesforce Marketing Cloudu koje služe za spremanje novih podatkovnih proširenja, stvaranje i organizaciju potrebnih podataka za ispravno korištenje same okoline uključujući i stvaranje datoteka za spremanje SQL upita. Primjer funkcije za stvaranje nove datoteke unutar Salesforce Marketing Clouda prikazan je na slici 4.25.

```
 * Creates a Folder
 * @param {Object} authConfig - Authorization Credentials.
 * @param {String} customerKey - CustomerKey of the Folder.
 * @param {String} name - Name of the Folder.
 * @param {String} description - Description of the Folder.
 * @param {String} contentType - ContentType of the Folder.
 * @param {Number} parentFolderId - Id of Parent Folder of the Folder.
 * @returns {Promise<Object>} - Returns created Folder.
 */
const create = (authConfig, customerKey, name, description, contentType, parentFolderId) => {
  let payload = `
    <soapenv:Body xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <CreateRequest
        xmlns="http://exacttarget.com/wsdl/partnerAPI">
        <Options/>
        <ns1:Objects
            xmlns:ns1="http://exacttarget.com/wsdl/partnerAPI"
            xsi:type="ns1:DataFolder">
            <ns1:ModifiedDate
             xsi:nil="true"/>
            <ns1:ObjectID
             xsi:nil="true"/>
            <ns1:CustomerKey>${customerKey}</ns1:CustomerKey>`;
  if (parentFolderId) {
    payload += `
        <ns1:ParentFolder>
            <ns1:ID>${parentFolderId}</ns1:ID>
        </ns1:ParentFolder>`;
  }

  payload += `
            <ns1:Name>${name}</ns1:Name>
            <ns1:Description>${description}</ns1:Description>
            <ns1:ContentType>${contentType}</ns1:ContentType>
            <ns1:IsActive>true</ns1:IsActive>
            <ns1:IsEditable>true</ns1:IsEditable>
            <ns1:AllowChildren>true</ns1:AllowChildren>
        </ns1:Objects>
      </CreateRequest>
    </soapenv:Body>`;

  return new Promise((resolve, reject) => {
    soap.execute(authConfig, Constants.SFMC_SOAP_ACTION__CREATE, Constants.SFMC_OBJECT__DATA_FOLDER, payload, (error,
      data) => {
      if (error) {
        Util.handleError(error, reject);
      } else if (data && data['soap:Fault']) {
        error = Util.getErrorMessageFromSoapFault(data['soap:Fault']);
        Util.handleError(error, reject);
      } else if (data && data.Results && data.OverallStatus === 'Error') {
        Util.handleError(data.Results.StatusMessage, reject);
      } else {
        const newFolderId = data && data.Results && data.Results.NewID ? data.Results.NewID : null;
        const folder = data.Results;
        resolve({ folder, newFolderId });
      }
    });
  });
};
```

Sl. 4.25. Prikaz funkcije za stvaranje nove datoteke

Prema slici 4.25, vidljivo je kako funkcija create() prima sljedeće parametre: authConfig (vjerodajnice za provjeru autentičnosti), customerKey (jedinstveni identifikator datoteke), name (ime datoteke), description (opis datoteke), contentType (tip datoteke), parentFolderId (jedinstveni identifikator roditeljske datoteke). U funkciji se kreira tijelo SOAP API-ja [24] koji se izvršava funkcijom execute() te se nakon izvršavanja provjerava ukoliko je došlo do pogreške ili su podaci ispravno dohvaćeni.

- Rukovanje SQL aktivnostima unutar Salesforce Marketing Clouda što predstavlja jedno od najbitnijih funkcionalnosti u integraciji sa Salesforce Marketing Cloudom. U

SQL aktivnostima sadržani su svi upiti koji su stvoreni koristeći okolinu razvijenu u ovom diplomskom radu. Unutar SQL aktivnosti se spremaju SQL upiti koji se izvršavaju unutar Salesforce Markting Clouda te vraćaju rezultate. Prije nego što se SQL upiti pokrene potrebno je stvoriti novu SQL aktivnost funkcijom prikazanoj na slici 4.26. Prema slici 4.26 vidljivo je kako funkcija create() prima sljedeće parametre: authConfig (vjerodajnice za provjeru autentičnosti), settings (postavke SQL aktivnosti). Na početku se pomoću funkcije excapeXML() očiste specijalni XML znakove kao što su &, <, > te se pomoću funkcije covertDataActionKeyWord() definiraju radnje nad podacima SQL aktivnosti. Radnje nad podacima SQL aktivnosti određuju kako će podaci, dobiveni nakon izvršavanja SQL aktivnosti, biti upisani u podatkovno proširenje, a to su: dodati (engl. append), ažurirati (engl. update) i prepisati (engl. overwrite). Na kraju se u funkciji create() stvara tijelo SOAP API-ja [25] koji se izvršava funkcijom execute() te se nakon izvršavanja provjerava ukoliko je došlo do pogreške ili su podaci ispravno dohvaćeni. Nakon što se stvori nova SQL aktivnost SQL upit se pokreće pomoću funkcije prikazane na slici 4.27. Prema slici 4.27, vidljivo je kako funkcija run() prima sljedeće parametre: authConfig (vjerodajnice za provjeru autentičnosti), objectId (jedinstveni identifikator SQL aktivnosti koja se pokreće). Najprije se u funkciji create() stvara tijelo SOAP API-ja koji se izvršava funkcijom execute() te se nakon izvršavanja provjerava ukoliko je došlo do pogreške ili su podaci ispravno dohvaćeni.

```
/**
 * Creates a new QueryActivity.
 * @param {Object} authConfig - Authorization Credentials.
 * @param {Object} settings - Properties of the QueryActivity.
 * @returns {Promise<String>} - ObjectId of created QueryActivity.
 */
const create = (authConfig, settings) => {
  const name = xml.escapeXML(settings.name || `sfmc_${new Date().getTime()}`);
  const description = xml.escapeXML(settings.description);
  const query = xml.escapeXML(settings.query);
  const dataAction = Util.convertDataActionKeyword(settings.dataAction);

  let payload = `
    <soapenv:Body xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <CreateRequest xmlns="http://exacttarget.com/wsdl/partnerAPI">
       <Options>
       </Options>
       <Objects xsi:type="QueryDefinition">
        <PartnerKey xsi:nil="true"></PartnerKey>
        <ObjectID xsi:nil="true"></ObjectID>
        <Name>${name}</Name>
        <Description>${description}</Description>
        <QueryText>${query}</QueryText>
        <TargetType>DE</TargetType>`;

  if (settings.folderId) {
    payload += `        <CategoryID>${settings.folderId}</CategoryID>\n`;
  }

  payload += `
        <DataExtensionTarget>
         <CustomerKey>${settings.DECustomerKey}</CustomerKey>
        </DataExtensionTarget>
        <TargetUpdateType>${dataAction}</TargetUpdateType>
       </Objects>
      </CreateRequest>
     </soapenv:Body>`;

  return new Promise((resolve, reject) => {
    soap.execute(authConfig, Constants.SFMC_SOAP_ACTION__CREATE, Constants.SFMC_OBJECT__QUERY_DEFINITION, payload,
      (error, data) => {
      if (error) {
        Util.handleError(error, reject);
      } else if (
        data.Results &&
        data.Results.NewObjectID &&
        typeof data.Results.NewObjectID !== 'undefined'
      ) {
        // New query activity was successfully created

        if (data && data.Results && data.OverallStatus) {
          resolve(data.Results.Object);
        } else {
          Util.handleError('Unknown error from SFMC API', reject);
        }
        // New QueryActivity creation failed
      } else if (data.OverallStatus === 'Error') {
        const error =
            data && data.Results && data.Results.StatusMessage ?
              data.Results.StatusMessage :
              'Unknown error from SFMC API';
        reject(Util.cleanErrorMessage(error));
        Util.handleError(error, reject);
      } else if (data && data['soap:Fault']) {
        const error = Util.getErrorMessageFromSoapFault(data['soap:Fault']);
        Util.handleError(error, reject);
      } else {
        Util.handleError('Unexpected error', reject);
      }
    });
  });
};
```

Sl. 4.26. Prikaz funkcije za stvaranje SQL aktivnosti

```
/* Starts execution of a queryActivity. Returns a cb(error, task) */
/**
 * Runs the given QueryActivity
 * @param {Object} authConfig - Authorization Credentials.
 * @param {String} objectId - ObjectId of the QueryActivity to run.
 * @returns {Promise<Object>} - Task of the QueryActivity.
 */
const run = (authConfig, objectId) => {
  const payload = `
  <soapenv:Body xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <PerformRequestMsg xmlns="http://exacttarget.com/wsdl/partnerAPI">
     <Action>Start</Action>
     <Definitions>
      <ns1:Definition xmlns:ns1="http://exacttarget.com/wsdl/partnerAPI" xsi:type="ns1:QueryDefinition">
       <ns1:PartnerKey xsi:nil="true"/>
       <ns1:ModifiedDate xsi:nil="true"/>
       <ns1:ObjectID>${objectId}</ns1:ObjectID>
      </ns1:Definition>
     </Definitions>
    </PerformRequestMsg>
   </soapenv:Body>`;

  return new Promise((resolve, reject) => {
    soap.execute(authConfig, Constants.SFMC_SOAP_ACTION__PERFORM, Constants.SFMC_OBJECT__QUERY_DEFINITION, payload,
      (error, data) => {
      if (error) {
        Util.handleError(error, reject);
      } else if (data && data.Results) {
        if (data.Results.Result.StatusCode === 'OK') {
          resolve(data.Results.Result ? data.Results.Result.Task : null);
        } else {
          if (data.Results.StatusMessage) {
            Util.handleError(data.Results.StatusMessage, reject);
          } else {
            Util.handleError(data.Results.Result.StatusMessage, reject);
          }
        }
      } else if (data && data['soap:Fault']) {
        const error = Util.getErrorMessageFromSoapFault(data['soap:Fault']);
        Util.handleError(error, reject);
      } else {
        Util.handleError('Unknown error from SFMC API', reject);
      }
    });
  });
};
```

Sl. 4.27. Prikaz funkcije za pokretanje SQL aktivnosti

Prilikom procesa izvršavanja SQL upita mogu se dobiti trenutni statusi: U redu, obrada, dovršeno i pogreška kako bi korisnik znao u kojem je procesu pokrenuta SQL aktivnost. Prikaz funkcije za dohvaćanje trenutnog statusa SQL aktivnosti vidljiv je na slici 4.28. Prema slici 4.28. vidljivo je kako funkcija status() prima sljedeće parametre: authConfig (vjerodajnice za provjeru autentičnosti), taskId (jedinstveni identifikator pokrenute SQL aktivnosti). Najprije se u funkciji create() stvara tijelo SOAP API-ja [26] koji se izvršava funkcijom execute() te se nakon izvršavanja provjerava ukoliko je došlo do pogreške ili su podaci ispravno dohvaćeni.

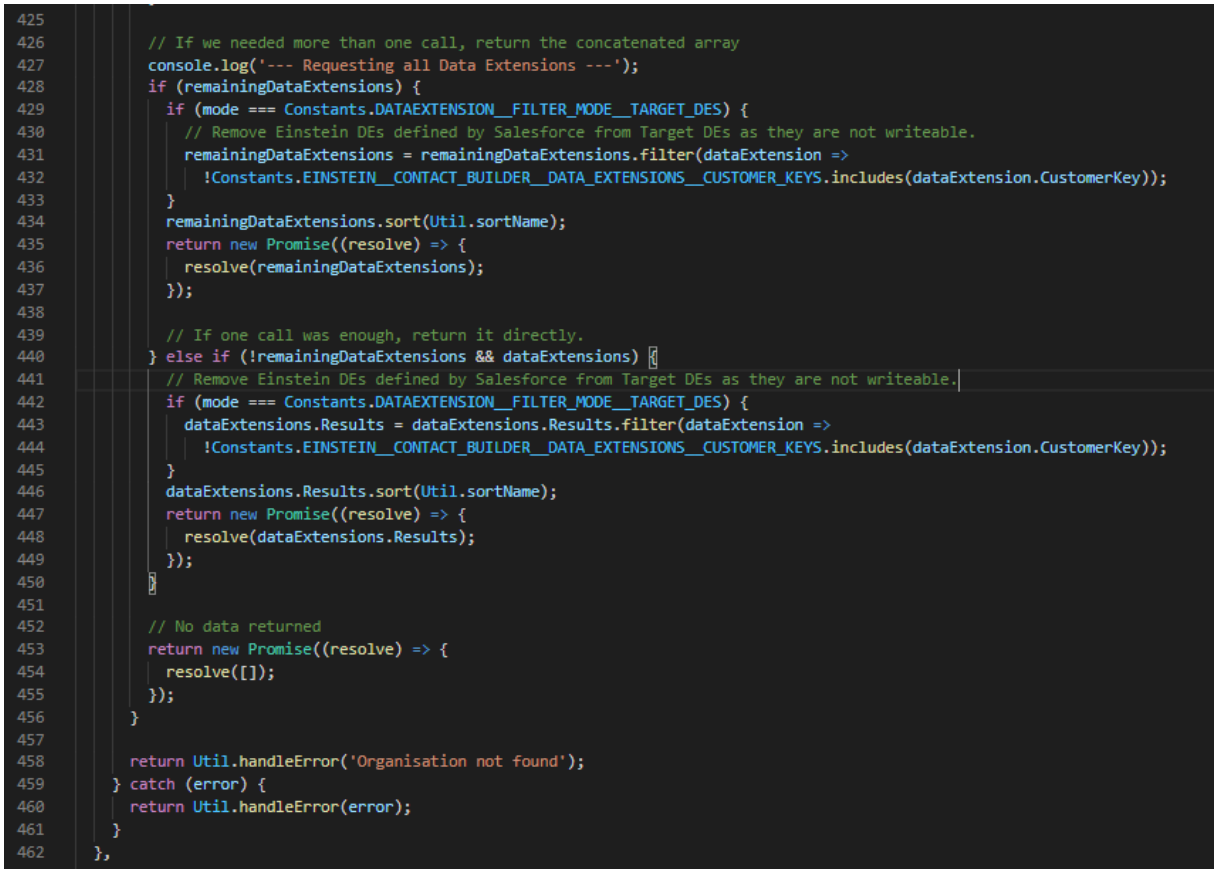
```
/**
 * Gets the status of a QueryActivity task.
 * Status can be one of the following: Queued, Processing, Complete, Error.
 * @param {Object} authConfig - Authorization Credentials.
 * @param {String} taskId - TaskId of the QueryActivity.
 * @returns {Promise<Object>} - Returns results.
 */
const status = (authConfig, taskId) => {
  const payload = `<soapenv:Body>
    <RetrieveRequestMsg xmlns="http://exacttarget.com/wsdl/partnerAPI">
      <RetrieveRequest>
        <ObjectType>AsyncActivityStatus</ObjectType>
        <Properties>Status</Properties>
        <Properties>StartTime</Properties>
        <Properties>EndTime</Properties>
        <Properties>TaskID</Properties>
        <Properties>ParentInteractionObjectID</Properties>
        <Properties>InteractionID</Properties>
        <Properties>Program</Properties>
        <Properties>StepName</Properties>
        <Properties>ActionType</Properties>
        <Properties>Type</Properties>
        <Properties>Status</Properties>
        <Properties>CustomerKey</Properties>
        <Properties>ErrorMsg</Properties>
        <Properties>CompletedDate</Properties>
        <Properties>StatusMessage</Properties>
        <Filter xsi:type="SimpleFilterPart">
          <Property>TaskID</Property>
          <SimpleOperator>equals</SimpleOperator>
          <Value>${taskId}</Value>
        </Filter>
      </RetrieveRequest>
    </RetrieveRequestMsg>
  </soapenv:Body>`;

  return new Promise((resolve, reject) => {
    soap.execute(authConfig, Constants.SFMC_SOAP_ACTION__RETRIEVE, Constants.SFMC_OBJECT__QUERY_DEFINITION, payload,
      async (error, data) => {
      try {
        // Handling errors and no Results
        data = await Util.handleRetrieveResponse(error, data, reject);

        if (data.Results && data.Results.length && data.Results[0].Properties && data.Results[0].Properties.Property) {
          const result = {
            moment: new Date().getTime(),
          };
          for (let i = 0; i < data.Results[0].Properties.Property.length; i += 1) {
            const name = data.Results[0].Properties.Property[i].Name;
            if (
              name === 'CompletedDate' ||
              name === 'StatusMessage' ||
              name === 'Status' ||
              name === 'ErrorMsg'
            ) {
              result[name] = data.Results[0].Properties.Property[i].Value;
            }
          }
          resolve(result);
        } else {
          // OverallStatus may contain error message
          Util.handleError(`Unknown error from SFMC API: ${
            data[0] ? data[0].OverallStatus : 'Make sure you provided a valid TaskID.'
          }`, reject);
        }
      } catch (error) {
        return Util.handleError(error, reject);
      }
    });
  });
};
```

Sl. 4.28. Prikaz funkcije za dohvaćanje statusa SQL aktivnosti

Nakon što su stvorene funkcije za integraciju sa Salesforce Marketing Cloudom razvijena je logika stvaranja složenih SQL upita. Pri stvaranju složenih SQL upita korištene su različite tehnike i mogućnosti samog SQL jezika od povezivanja izvorišnih podatkovnih proširenja do stvaranja raznovrsnih filtera. Kako bi to bilo omogućeno prvo je potrebno dohvatiti sva dostupna podatkovna proširenja. Nakon što se dohvate dostupna podatkovna proširenja korisnik odabire izvorišno podatkovno proširenje kao početni skup podataka koji će poslužiti za stvaranje SQL upita. Funkcija za dohvaćanje svih dostupnih podatkovnih proširenja prikazana je na slikama 4.29 i 4.30.

```
/**
 * Retrieves all Data Extensions for the given Organisation. Result is sorted by Name.
 * DESelect Folder is excluded from the search by default.
 * @param {Object} user - User Details.
 * @param {Object[]} remainingDataExtensions - Used for Continuous Calls.
 * @param {String} requestId - ID of the Request for Continuous Calls.
 * @param {String} mode - Mode parameter for Inclusion-Exclusion of selected Folders.
 * @param {Object} savedFilter - Used to save created Filter, and use it on Continuous Calls.
 * @param {Boolean} retrieveAllProperties - If true, retrieves Relation/Retention settings as well.
 * @param {Boolean} queryAllAccounts - Queries all business units.
 * @returns {Promise<Object[]>} - Array of Data Extensions.
 */
getAllDataExtensionsForOrg: async (user, remainingDataExtensions, requestId,
  mode, savedFilter, retrieveAllProperties, queryAllAccounts) => {
  try {
    const oauthToken = await User.checkAccessToken(user);
    const org = await Org.findOne({ _id: user.orgId });

    // Use the helper function to create filter using Mode property
    const filter = savedFilter || await DataExtension.createFilterToRetrieveDataExtensions(user, mode);

    if (org) {
      let dataExtensions;
      if (env.USE_SFMC_MOCKUP) {
        dataExtensions = sfmcMock.dataExtensions.list;
      } else {
        dataExtensions = await sfmc.dataExtensions.list(
          {
            oauthToken,
            subdomain: org.marketingCloudSubdomain,
          },
          filter,
          requestId || null,
          retrieveAllProperties,
          queryAllAccounts,
        );
      }

      // See if we already made a call, and concatenate results
      if (remainingDataExtensions) {
        remainingDataExtensions = remainingDataExtensions.concat(dataExtensions.Results);
      }

      // If one call was not enough to retrieve all DEs, save what we retrieved so far,
      // and call the same function again. OverallStatus indicates if there are more data to retrieve or not.
      // RequestID is used to retrieve rest of the available data, from the first call.
      if (dataExtensions && dataExtensions.OverallStatus === 'MoreDataAvailable') {
        return new Promise((resolve) => {
          resolve(DataExtension.getAllDataExtensionsForOrg(
            user,
            remainingDataExtensions || dataExtensions.Results,
            dataExtensions.RequestID,
            mode,
            filter,
            retrieveAllProperties,
            true,
          ));
        });
      }
```

Sl. 4.29. Prikaz funkcije za dohvaćanje podatkovnih proširenja (A)



```
425
426       // If we needed more than one call, return the concatenated array
427       console.log('--- Requesting all Data Extensions ---');
428       if (remainingDataExtensions) {
429         if (mode === Constants.DATAEXTENSION__FILTER_MODE__TARGET_DES) {
430           // Remove Einstein DEs defined by Salesforce from Target DEs as they are not writeable.
431           remainingDataExtensions = remainingDataExtensions.filter(dataExtension =>
432             !Constants.EINSTEIN__CONTACT_BUILDER__DATA_EXTENSIONS__CUSTOMER_KEYS.includes(dataExtension.CustomerKey));
433         }
434         remainingDataExtensions.sort(Util.sortName);
435         return new Promise((resolve) => {
436           resolve(remainingDataExtensions);
437         });
438
439         // If one call was enough, return it directly.
440       } else if (!remainingDataExtensions && dataExtensions) {
441         // Remove Einstein DEs defined by Salesforce from Target DEs as they are not writeable.
442         if (mode === Constants.DATAEXTENSION__FILTER_MODE__TARGET_DES) {
443           dataExtensions.Results = dataExtensions.Results.filter(dataExtension =>
444             !Constants.EINSTEIN__CONTACT_BUILDER__DATA_EXTENSIONS__CUSTOMER_KEYS.includes(dataExtension.CustomerKey));
445         }
446         dataExtensions.Results.sort(Util.sortName);
447         return new Promise((resolve) => {
448           resolve(dataExtensions.Results);
449         });
450       }
451
452       // No data returned
453       return new Promise((resolve) => {
454         resolve([]);
455       });
456     }
457
458     return Util.handleError('Organisation not found');
459   } catch (error) {
460     return Util.handleError(error);
461   }
462 },
```

Sl.4.30. Prikaz funkcije za dohvaćanje podatkovnih proširenja (B)

Prema slikama 4.29 i 4.30, vidljivo je kako funkcija getAllDataExtensionsForOrg() prima sljedeće parametre: user (podaci o korisniku), remainingDataExtensions (služi za ponovo dohvaćanje podatkovnih proširenja ukoliko nisu sva već dohvaćena), requestId (jedinstveni identifikator za ponovno dohvaćanje podatkovnih proširenja), mode (određuje dohvaćaju li se izvorišna ili ciljna podatkovna proširenja), savedFilter (služi za filtriranje dohvata podatkovnih proširenja iz datoteka Salesforce Marketing Clouda), retrieveAllProperties (određuje hoće li se dohvatiti sva svojstva podatkovnih proširenja), queryAllAcounts (određuje hoće li se dohvatiti podatkovna proširenja iz svih korisničkih računa). Najprije se pomoću funkcije checkAccessToken() provjerava da li je pristupni token korisnika i dalje valjan, ako nije, osvježava se te se korisnik ažurira. Zatim se, pomoću funkcije findOne(), dohvaća organizacija iz baze podataka te se pomoću funkcije createFilterToRetrieveDataExtensions() stvaraju filtri za filtriranje podatkovnih proširenja po pojedinim datotekama Salesforce Marketing Clouda. Nadalje, ukoliko postoji organizacija pomoću funkcije list() dohvaća se lista svih dostupnih podatkovnih proširenja te ukoliko vraćena vrijednost ukazuje na to da nisu sva podatkovna proširenja dohvaćena ponovo se poziva funkcija getAllDataExtensionsForOrg() dok se ne dohvate sva podatkovna proširenja.

Na kraju funkcija sortira sva podatkovna proširenja te ih vraća kao povratnu vrijednost. Nakon što korisnik odabere željeno podatkovno proširenje korisniku je omogućeno definiranje različitih vrsta filtriranja koje je moguće obaviti nad podacima izvorišnih podatkovnih proširenja. Filtriranje predstavlja jedan od temeljnih funkcionalnosti okoline jer omogućavaju stvaranje različitih SQL upita. Nakon odabira izvorišnog podatkovnog proširenja te definiranja filtera potrebno je odabrati ciljno podatkovno proširenje u koje će podaci dobiveni SQL upitom biti upisani. Postupak odabira ciljnog podatkovnog proširenja može se uraditi na dva načina. Prvi način je odabir već postojećeg podatkovnog proširenja. Ukoliko korisnik odabere već postojeće podatkovno proširenje poziva se funkcija getFields() prikazana na slikama 4.31 i 4.32 kako bi se dohvatilo željeno podatkovno proširenje s pripadajućim poljima. Prema slikama 4.31 i 4.32 vidljivo je kako funkcija getFields() prima sljedeće parametre: user (podaci o korisniku), customerKey (jedinstveni identifikator podatkovnog proširenja čija se polja dohvaćaju), objectId (jedinstveni identifikator polja ukoliko se želi dohvatiti samo jedno polje podatkovnog proširenja), dataExtensionObject (podatkovno proširenje čija se polja dohvaćaju), dataExtensionContentMap (tip podatkovnog proširenja). Najprije se pomoću funkcije checkAccessToken() provjerava da li je pristupni token korisnika i dalje valjan, ako nije, osvježava se te se korisnik ažurira. Zatim se, pomoću funkcije findOne(), dohvaća organizacija iz baze podataka. Nadalje, provjerava se da li je tip podatkovnog proširenja, čija polja se dohvaćaju, već poznat. Tip podatkovnog proširenja potreban je iz razloga što se polja pojedinih podatkovnih proširenja moraju pretraživati kroz sve korisničke račune. Funkcijom list() dohvaćaju se sva podatkovna proširenja kako bi se omogućilo pronalazak tipa podatkovnog proširenja pomoću funkcije getContentType(). Ovisno o tipu podatkovnog proširenja funkcijom checkIsSharedDataExtension() određuje se da li je potrebno pretražiti i dohvatiti polja iz svih korisničkih računa za željeno podatkovno proširenje. Nakon uspješnog postavljanja tipa podatkovnog proširenja provjerava se da li postoji organizacija te ako postoji pomoću funkcije list() dohvaćaju se polja podatkovnog proširenja. Na kraju, ukoliko nije došlo do nikakve greške funkcija vraća polja podatkovnog proširenja te tip podatkovnog proširenja.

```
/**
 * Retrieve Fields of a Data Extension by given CustomerKey.
 * @param {Object} user - User Details.
 * @param {String} customerKey - CustomerKey of the Data Extension.
 * @param {String} objectId - ObjectID of the Field.
 * @param {Object} dataExtensionObject - Optional, can be given if dataExtension is already retrieved.
 * @param {Map} dataExtensionContentMap - If a contentType map is provided, use it to retrieve contentType.
 * @returns {Promise<Object[]>} - Object containing the Fields and Type of the Data Extension.
 */
getFields: async (user, customerKey, objectId, dataExtensionObject, dataExtensionContentMap) => {
  try {
    const oauthToken = await User.checkAccessToken(user);
    const org = await Org.findOne({ _id: user.orgId });

    // Do not query all accounts by default to avoid duplicate fields
    let queryAllAccounts = false;

    // Check if the call is made from child BU
    const isChildBU = user.loggedInBusinessUnitId !== org.enterpriseBusinessUnitId;

    // If contentType map is provided, find and assign the contentType
    let dataExtensionContentType;
    if (dataExtensionContentMap) {
      dataExtensionContentType = dataExtensionContentMap.get(customerKey);
    }

    // Data Extension's Folder
    let folder;

    // This if block checks if the DE is shared or not and if so, sets QAA true, furthermore,
    // retrieves the Data Extension's folder to find and return its type.
    if (!dataExtensionContentType && customerKey) {
      // Create a filter to retrieve given Data Extension.
      const settings = {
        simpleFilter: {
          property: Constants.SFMC__PROPERTY__CUSTOMER_KEY,
          operator: Constants.SFMC__OPERATOR__EQUALS,
          value: customerKey,
        },
      };

      // Define dataExtension
      let dataExtension = dataExtensionObject;
      // Retrieve Data Extension
      if (!dataExtension) {
        dataExtension = await sfmc.dataExtensions.list(
          {
            oauthToken,
            subdomain: org.marketingCloudSubdomain,
          },
          settings,
          null,
          false,
        );
        // Since we're calling with list, it's in an array. Take the first element.
        dataExtension = dataExtension.Results[0];
      }
```

Sl. 4.31. Prikaz funkcije za dohvaćanje polja podatkovnog proširenja (A)

```
308
309         // Retrieve Data Extension's Parent Folder
310         folder = await Folder.getFoldersByIds(
311           user,
312           [dataExtension.CategoryID]
313         );
314
315         // Since we're calling with list, it's in an array. Take the first element.
316         if (folder && Array.isArray(folder) && folder.length === 1) {
317           folder = folder[0];
318         }
319
320         // If contentType is not known but folder is retrieved, check its contentType to set QAA.
321         if (folder && isChildBU) {
322           // Sets true if contentType is one of the shared types.
323           queryAllAccounts = DataExtension.checkIsSharedDataExtension(folder.ContentType);
324         }
325
326       // If contentType is known and call is made from a child BU, check the contentType and set QAA accordingly.
327       } else if (dataExtensionContentType && isChildBU) {
328         // Retrieve the contentType from map and check if it's shared.
329         queryAllAccounts = DataExtension.checkIsSharedDataExtension(dataExtensionContentType);
330       }
331
332       // Retrieve fields.
333       if (org) {
334         let fields;
335         if (env.USE_SFMC_MOCKUP) {
336           fields = sfmcMock.dataExtensions.fields(customerKey);
337         } else {
338           fields = await sfmc.dataExtensions.fields(
339             {
340               oauthToken,
341               subdomain: org.marketingCloudSubdomain,
342             },
343             customerKey,
344             objectId,
345             queryAllAccounts,
346           );
347         }
348         if (fields) {
349           return new Promise((resolve) => {
350             resolve({ fields, contentType: folder ? folder.ContentType : dataExtensionContentType });
351           });
352         }
353
354         // No data returned
355         return new Promise((resolve) => {
356           resolve([]);
357         });
358       }
359
360       return Util.handleError('Organisation not found');
361     } catch (error) {
362       return Util.handleError(error);
363     }
364   },
```

Sl. 4.32. Prikaz funkcije za dohvaćanje polja podatkovnog proširenja (B)

Ukoliko korisnik želi stvoriti novo prilagođeno podatkovno proširenje umjesto već postojećih poziva se funkcija prikazana na slici 4.33 za stvaranje novog podatkovnog proširenja s odgovarajućim poljima.

```
/**
 * Creates a new Data Extension Object.
 * @param {Object}user - User Details.
 * @param {String}name - Name of Data Extension.
 * @param {Object}description - Description for Data Extension.
 * @param {Object[]}fields - Fields of Data Extension.
 * @param {Number}folderId - Parent Folder's ID to create Data Extension in.
 * @param {Object}retentionPolicy - Retention settings of Data Extension.
 * @param {Object}relationship - Relationship settings of Data Extension.
 * @param {String}customerKey - CustomerKey of the Data Extension.
 * @returns {Promise<Object>} - Created Data Extension Object.
 */
create: async (user, name, description, fields, folderId, retentionPolicy, relationship, customerKey) => {
  console.log(
    `creating data extension with user ${JSON.stringify(
      user,
    )} name: ${name} fields: ${JSON.stringify(fields)}`,
  );

  if (name && fields) {
    try {
      const oauthToken = await User.checkAccessToken(user);
      const org = await Org.findOne({ _id: user.orgId });
      if (org) {
        if (!folderId) {
          // get deselectDataExtensionsFolderId, create new folder if none exists yet
          folderId = await Folder.getOrCreateDataExtensionsFolder(
            org,
            user,
          );
        }

        const newDataExtension = await sfmc.dataExtensions.create(
          {
            oauthToken,
            subdomain: org.marketingCloudSubdomain,
          },
          {
            name,
            description,
            fields,
            folderId,
            customerKey,
          },
          retentionPolicy || null,
          relationship || null,
        );
        console.log('DE successfully created');

        return new Promise((resolve) => {
          resolve(newDataExtension);
        });
      }

      return Util.handleError('Organisation not found');
    } catch (error) {
      return Util.handleError(error);
    }
  } else {
    return Util.handleError('Data Extension has no name or fields');
  }
},
```

Sl. 4.33. Prikaz funkcije za stvaranje novog podatkovnog proširenja

Prema slici 4.33 vidljivo je kako funkcija create() prima sljedeće parametre: user (podaci o korisniku), name (ime podatkovnog proširenja), description (opis podatkovnog proširenja), fields (polja podatkovnog proširenja), folderId (jedinstveni identifikator datoteke u kojoj će

biti spremljeno novo podatkovno proširenje), retentionPolicy (postavke zadržavanja), relationship (postavke odnosa podatkovnog proširenja) i customerKey (jedinstveni identifikator podatkovnog proširenja). Najprije se provjerava da li postoje ime i polja podatkovnog proširenja koje se želi kreirati. Ukoliko ne postoje funkcija će vratiti grešku s porukom o nepostojanju imena i polja podatkovnog proširenja. Ukoliko ime i polja postoje pomoću funkcije checkAccessToken() provjerava se da li je pristupni token korisnika i dalje valjan, ako nije, osvježava se te se korisnik ažurira. Zatim se, pomoću funkcije findOne(), dohvaća organizacija iz baze podataka. Ukoliko je organizacija pronađena provjerava se postoji li parametar folderId. Ukoliko ne postoji pomoću funkcije getOrCreateDataExtensionsFolder() kreira se nova datoteka unutar Salesforce Markting Clouda u kojoj će biti spremljeno novo podatkovno proširenje. Nakon toga se kreira novo podatkovno proširenje funkcijom dataExtensions.create(). Ukoliko nije došlo do pogreške funkcija create() će vratiti novo podatkovno proširenje. Nakon odabira izvorišnog podatkovnog proširenja, definiranja filtera te odabir ciljnog podatkovnog proširenja stvara se SQL upit. Najprije se definirao početni dio SQL upita odnosno klauzula SELECT koja određuje polja podatkovnog proširenja čije vrijednosti će biti prikazane nakon izvršenog SQL upita. Zbog mogućnosti odabira polja iz više podatkovnih proširenja za stvaranje SQL upita klauzule SELECT koristili su se pseudonimi koji predstavljaju polja ciljnog podatkovnog proširenja [27]. Dio funkcije za stvaranje klauzule SELECT SQL upita prikazan je na slici 4.34.

```
1279            fieldStrings.push(
1280              `"${Query.escapeSQLCharacters(fieldsArray[f].collectionAlias, true)}"."${fieldsArray[f].field}" AS "$
                  {fieldsArray[f].alias}"`
1281            );
```

Sl. 4.34. Prikaz dijela funkcije za stvaranje SELECT klauzule

Nakon što se definirala klauzula SELECT SQL upita potrebno je definirati sljedeću klauzulu koja navodi kojem podatkovnom proširenju pripadaju polja korištena u klauzuli SELECT. Klauzula u kojem se navodi izvorna podatkovna proširenja iz kojih se podaci čitaju je klauzula FROM. U klauzuli FROM se definiraju i odnosi između podatkovnih proširenja [28]. Prikaz dijela funkcije za stvaranje klauzule FROM vidljiv je na slici 4.35.

```
// Add FROM + JOINS
query += ' FROM ';
// Create WHERE xx = null for LEFT_WITHOUT and RIGHT_WITHOUT joins
const joinWithoutWhereClauseArray = [];

if (relations.length === 0 && collections.length === 1) {
  // Check if the Collection's Name starts with an underscore. Underscore means it's a DataView,
  // and DataViews work only when called without quotes around them.
  const contentType = dataExtensionsContentTypesMap.get(collections[0].collectionCustomerKey);
  const collectionName = Query.getValidCollectionName(collections[0].collection,
    isQueryRunByChildBusinessUnit, contentType);
  query += `${collectionName} "${Query.escapeSQLCharacters(collections[0].alias, true)}"`;
} else {
  for (let r = 0; r < relations.length; r += 1) {
    if (r === 0) {
      const contentType = dataExtensionsContentTypesMap.get(relations[r].fromCollectionCustomerKey);
      const collectionName = Query.getValidCollectionName(relations[r].fromCollection,
        isQueryRunByChildBusinessUnit, contentType);
      query += `${collectionName} "${Query.escapeSQLCharacters(relations[r].fromCollectionAlias, true)}"`;
    }
    // add space before the join
    query += ' ';
    switch (relations[r].type) {
      case Constants.RELATIONTYPE__LEFT_WITHOUT:
        query += 'LEFT';
        joinWithoutWhereClauseArray.push(
          ` "${Query.escapeSQLCharacters(relations[r].toCollectionAlias, true)}"."${relations[r].toField}" IS NULL`
        );
        break;

      case Constants.RELATIONTYPE__RIGHT_WITHOUT:
        query += 'RIGHT';
        joinWithoutWhereClauseArray.push(
          ` "${Query.escapeSQLCharacters(relations[r].fromCollectionAlias, true)}"."${relations[r].fromField}" IS
          NULL`
        );
        break;

      default:
        query += relations[r].type;
        break;
    }
    const contentType = dataExtensionsContentTypesMap.get(relations[r].toCollectionCustomerKey);
    const collectionName = Query.getValidCollectionName(relations[r].toCollection, isQueryRunByChildBusinessUnit,
      contentType);
    query +=
      ` JOIN ${collectionName} "${
        Query.escapeSQLCharacters(relations[r].toCollectionAlias, true)}" ON "${
        Query.escapeSQLCharacters(relations[r].fromCollectionAlias, true)
      }"."${relations[r].fromField}" = "${Query.escapeSQLCharacters(relations[r].toCollectionAlias, true)}"."${
        relations[r].toField
      }"`;
  }
```

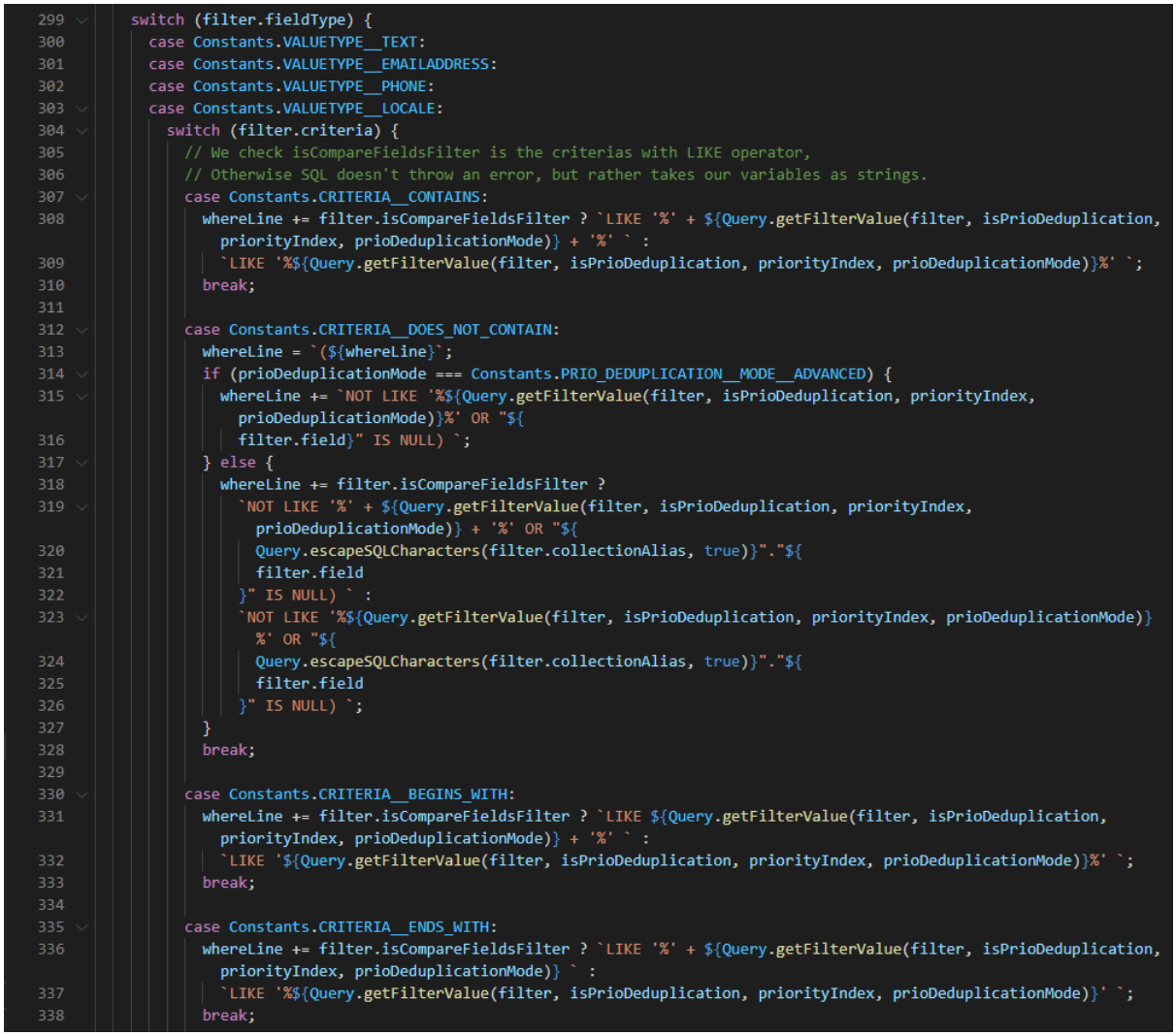
Sl. 4.35. Prikaz dijela funkcije za stvaranje FROM klauzule

Prema slici 4.35 vidljivo je kako se prvo na postojeći upit dodaje klauzula FROM nakon čega se provjerava da li se je odabrano samo jedno ciljno proširenje ili je odabrano više podatkovnih proširenja. Ukoliko je odabrano samo jedno podatkovno proširenje dohvaća se njegovo ime i njegov pseudonim te se čiste posebni znakovi poput jednostrukih navodnika i zareza. Ukoliko je odabrano više podatkovnih proširenja for petljom se prolazi kroz njihove odnose te se zavisno o tipu odnosa (join) stvaraju naredbe poput RIGHT, LEFT, INNER JOIN i OUTER JOIN. Nakon toga se stvara dio klauzule FROM koja se odnosi na polja podatkovnih proširenja kojima su povezani. Nakon što se definirala klauzula FROM slijedi klauzula WHERE koja služi za definiranje kriterija za ograničavanje sadržaja konačnog

rezultata SQL upita. Klauzula WHERE predstavlja najsloženiji dio stvaranja SQL upita zbog toga što postoje razni pod-dijelovi klauzule WHERE i operatori. Operatori koje se mogu upotrijebiti kao dio klauzule WHERE te koji su implementirane u ovoj okolinu su: jednako (=) , veće od (>) , manje od (<), veće od ili jednako (>=), manje od ili jednako (<=), različito (<>), BETWEEN, LIKE, IN. Osim navedenih operatora klauzula WHERE omogućuje i stvaranje podupita te korištenje formula kao što su: Count, Average, Sum, Minimum, Maximum uz pomoć klauzula HAVING i GROUP BY. Dijelovi funkcije korištenih za ostvarivanje operatora klauzule WHERE prikazani su na slikama 4.36 i 4.37.

```
253     switch (filter.criteria) {
254       case Constants.CRITERIA__EQUALS:
255       case Constants.CRITERIA__NOT_EQUAL_TO:
256       case Constants.CRITERIA__SMALLER_THAN:
257       case Constants.CRITERIA__SMALLER_THAN_OR_EQUAL_TO:
258       case Constants.CRITERIA__GREATER_THAN:
259       case Constants.CRITERIA__GREATER_THAN_OR_EQUAL_TO:
260         whereLine += `${filter.criteria} `;
261         break;
262
263       // For Contains, Does not contain, begins with and Does not begin with: see switch fieldType
264       case Constants.CRITERIA__IS_EMPTY:
265         whereLine += 'IS NULL ';
266         break;
267
268       case Constants.CRITERIA__IS_NOT_EMPTY:
269         whereLine += 'IS NOT NULL ';
270         break;
271
272       // See if the criteria is IN
273       case Constants.CRITERIA__IN:
274         whereLine += `IN (${Query.splitArrayToStrings(filter.value)}) `;
275         break;
276
277       // See if the criteria is NOT IN
278       case Constants.CRITERIA__NOT_IN:
279         whereLine += `NOT IN (${Query.splitArrayToStrings(filter.value)}) `;
280         break;
```

Sl. 4.36. Prikaz dijela funkcije za implementaciju operatora



```
switch (filter.fieldType) {
  case Constants.VALUETYPE__TEXT:
  case Constants.VALUETYPE__EMAILADDRESS:
  case Constants.VALUETYPE__PHONE:
  case Constants.VALUETYPE__LOCALE:
    switch (filter.criteria) {
      // We check isCompareFieldsFilter is the criterias with LIKE operator,
      // Otherwise SQL doesn't throw an error, but rather takes our variables as strings.
      case Constants.CRITERIA__CONTAINS:
        whereLine += filter.isCompareFieldsFilter ? `LIKE '%' + ${Query.getFilterValue(filter, isPrioDeduplication,
          priorityIndex, prioDeduplicationMode)} + '%' ` :
          `LIKE '%${Query.getFilterValue(filter, isPrioDeduplication, priorityIndex, prioDeduplicationMode)}%' `;
        break;

      case Constants.CRITERIA__DOES_NOT_CONTAIN:
        whereLine = `(${whereLine}`;
        if (prioDeduplicationMode === Constants.PRIO_DEDUPLICATION__MODE__ADVANCED) {
          whereLine += `NOT LIKE '%${Query.getFilterValue(filter, isPrioDeduplication, priorityIndex,
            prioDeduplicationMode)}%' OR "${
            filter.field}" IS NULL) `;
        } else {
          whereLine += filter.isCompareFieldsFilter ?
            `NOT LIKE '%' + ${Query.getFilterValue(filter, isPrioDeduplication, priorityIndex,
              prioDeduplicationMode)} + '%' OR "${
              Query.escapeSQLCharacters(filter.collectionAlias, true)}"."${
              filter.field
            }" IS NULL) ` :
            `NOT LIKE '%${Query.getFilterValue(filter, isPrioDeduplication, priorityIndex, prioDeduplicationMode)}
              %' OR "${
              Query.escapeSQLCharacters(filter.collectionAlias, true)}"."${
              filter.field
            }" IS NULL) `;
        }
        break;

      case Constants.CRITERIA__BEGINS_WITH:
        whereLine += filter.isCompareFieldsFilter ? `LIKE ${Query.getFilterValue(filter, isPrioDeduplication,
          priorityIndex, prioDeduplicationMode)} + '%' ` :
          `LIKE '${Query.getFilterValue(filter, isPrioDeduplication, priorityIndex, prioDeduplicationMode)}%' `;
        break;

      case Constants.CRITERIA__ENDS_WITH:
        whereLine += filter.isCompareFieldsFilter ? `LIKE '%' + ${Query.getFilterValue(filter, isPrioDeduplication,
          priorityIndex, prioDeduplicationMode)} ` :
          `LIKE '%${Query.getFilterValue(filter, isPrioDeduplication, priorityIndex, prioDeduplicationMode)}' `;
        break;
```

Sl. 4.37. Prikaz dijela funkcije za implementaciju LIKE operatora

Također je implementirana funkcionalnost za rukovanje s poljima koji sadrže datume. Korisniku je omogućeno filtriranje podataka na osnovu datuma koji se može odabrati ručno ili se mogu definirati godine, mjeseci, tjedni, dani, sati ili minute prije ili poslije trenutnog datuma pokretanja SQL upita. Dio funkcije koji implementira navedenu funkcionalnost prikazan je na slici 4.38.

```
438     case Constants.VALUETYPE__DATE:
439       switch (filter.dateFilterType) {
440         case Constants.SMART_DATE_TYPE__RELATIVE:
441           if (
442             filter.dateValueStart === Constants.RELATIVE_DATE_TYPE__BEFORE_TODAY ||
443             filter.dateValueStart === Constants.RELATIVE_DATE_TYPE__AFTER_TODAY
444           ) {
445             whereLine += `dateadd(${Query.getDateInterval(filter.filterInterval)}, ${
446               Query.convertNegativeIfBeforeToday(filter)}, CONVERT( DATE, getdate()) ) `;
447           } else if (
448             filter.dateValueStart === Constants.RELATIVE_DATE_TYPE__TODAY
449           ) {
450             whereLine += 'CONVERT( DATE, getdate() ) ';
451           } else {
452             whereLine += `dateadd(${Query.getDateInterval(filter.filterInterval)}, ${
453               Query.convertNegativeIfBeforeToday(filter)}, getdate()) `;
454           }
455           break;
```

Sl. 4.38. Prikaz dijela funkcije za rukovanje datumima

Nakon što se uspješno stvorio i zadnji dio SQL upita nastali upit se prosljeđuje funkcijama za pokretanje upita unutar Salesforce Marketing Clouda. Prvo se poziva funkcija za stvaranje nove SQL aktivnosti (Slika 4.39) te se nakon toga i pokreće (Slika 4.40).

```
538         // Create a new Query Activity
539         const queryActivity = await QueryActivity.createQueryActivity(
540           user,
541           customerKey,
542           query,
543           Util.removeIllegalCharacters(name) + ' ' + Date.now(),
544           `Query for ${selectionId}, automatically generated by Deselect`,
545           Constants.QUERYACTIVITY__TASKTYPE__RUN,
546           action,
547           selectionId,
548         );
549
550         // Assign it as ObjectID of QueryActivity.
551         queryActivityObjectID = queryActivity.ObjectID;
```

Sl. 4.39. Prikaz funkcije za stvaranje SQL aktivnosti

```
1248        const oauthToken = await User.checkAccessToken(user);
1249        const org = await Org.findOne({ _id: user.orgId });
1250
1251        if (org) {
1252          // Run the queryactivity in sfmc
1253          const task = await sfmc.queryActivity.run(
1254            {
1255              oauthToken,
1256              subdomain: org.marketingCloudSubdomain,
1257            },
1258            queryActivityObjectID
1259          );
1260
1261          const taskId = task.ID;
```

Sl. 4.40. Prikaz funkcije za pokretanje SQL aktivnosti

Na kraju je stvorena API POST metoda (Slika 4.41) kako bi klijentska strana mogla koristiti funkcije za stvaranje i pokretanje SQL upita.

```
15  /**
16   * Runs the QueryActivity of Run type, defined for the given Selection.
17   */
18  router.post('/query-activities/run', User.check, async (req, res) => {
19    const selectionId = req.body.selectionId;
20    const user = req.user;
21
22    try {
23      const runResult = await QueryActivity.createQueryAndRun(selectionId, user);
24      res.json({
25        taskId: runResult.taskId,
26        queryActivityObjectID: runResult.queryActivityObjectID,
27        success: true,
28      });
29    } catch (error) {
30      Util.handleApiErrorResponse(req, res, __filename, error);
31    }
32  });
```

Sl. 4.41. Prikaz API POST metoda za pokretanje SQL upita

Ukoliko nije došlo do nekakvih neočekivanih pogrešaka rezultati stvorenog SQL upita prosljeđuju se klijentskoj strani koja predstavlja sloj View u opisanoj MVC arhitekturi. Nadalje, klijentska strana obrađuje dobivene rezultate te ih reprezentira krajnjem korisniku.

4.2. Razvoj i način rada klijentske strane

Osnovni cilj klijentske strane je prikaz korisničkog sučelja koje će korisniku omogućavati stvaranje SQL upita. Korisniku je, pomoću „*drag and drop*“ metode, omogućeno korištenje funkcionalnosti okoline kako bi bez korištenja SQL jezika generirao dijelove za SQL upit. Klijentska strana komunicira s poslužiteljskom stranom kroz API-je što omogućava korištenje funkcionalnosti kao što su: povezivanje podatkovnih proširenja, stvaranje filtera i pod-upita, stvaranje novih i uređivanje postojećih podatkovnih proširenja. Na kraju klijentska strana prosljeđuje poslužiteljskoj strani bitne parametre za stvaranje SQL upita te dobivene rezultate prikazuje korisniku. U ovom poglavlju opisane su tehnologije i način stvaranja funkcionalnosti koje pruža korisničko sučelje.

4.2.1. Programski alati i tehnologije na klijentskoj strani

Klijentska strana razvijena je u okruženju Visual Studio Code koji služi za pisanje i uređivanje koda. Klijentska strana implementira korisničko sučelje koje je razvijeno u React.js okolini koristeći JavaScript programski jezik, HTML prezentacijski jezik te CSS opisni jezik. Kao posebni dodaci koriste se biblioteke: SweetAlert za prikazivanje poruka upozorenja i poruke pomoći, proptypes za validaciju svojstava komponenti, axios za komuniciranje s poslužiteljskom stranom te Salesforce Lightning Design biblioteka kako bi korisničko sučelje bilo dizajnirano što sličnije Salesforce Markting Cloud okolini.

4.2.1.1. HTML

HTML (HyperText Markup Language) najosnovniji je gradivni element internet stranica. Definira značenje i strukturu internet sadržaja te se obično kombinira sa CSS – om i JavaScript jezikom. " HyperText " odnosi se na poveznice koje međusobno povezuju internet stranice, bilo unutar jedne web stranice ili između web stranica [29]. HTML se koristi za označavanje teksta, slika i drugog sadržaja za prikaz u internet pregledniku. HTML oznaka uključuje posebne elemente kao što su <head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, , , < img>, <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <nav>, <output>, <progress>, <video>, <ul>, <ol> , <li> i mnogi drugi. HTML element odmaknut je od drugog teksta u dokumentu pomoću oznaka, koje se sastoje od imena elementa okruženog s "<" i ">". Ime elementa unutar oznake ne razlikuje velika i mala slova. Na primjer, oznaka <title> može se zapisati kao <Title>, <TITLE> ili na bilo koji drugi način [30].

4.2.1.2. CSS

Cascading Style Sheets (CSS) jezik je stilskih tablica koji se koristi za opis prezentacije dokumenta napisanog u HTML-u ili XML-u. CSS opisuje kako se elementi trebaju prikazivati na zaslonu ili na drugim medijima. CSS je među osnovnim jezicima otvorenog interneta i standardiziran je u svim internet preglednicima prema W3C specifikaciji. Prethodno se razvoj različitih dijelova CSS specifikacije odvijao sinkrono, što je omogućavalo upotrebu najnovije verzije CSS-a. Od CSS3 verzije, opseg specifikacije znatno se povećava, a napredak na različitim CSS modulima raste sve više i više [31].

4.2.1.3. JavaScript

Iako je najpoznatiji kao skriptni jezik za internet stranice, koriste ga i mnoga okruženja koja nisu preglednici, poput Node.js, Apache CouchDB i Adobe Acrobat, React.js, itd. JavaScript je jednonitni, dinamični jezik, koji podržava stilove objektno orijentiranog i funkcionalnog programiranja [32]. Standard za JavaScript je ECMAScript. Od 2012. godine svi moderni preglednici u potpunosti podržavaju ECMAScript 5.1. Stariji preglednici podržavaju barem ECMAScript 3. 2015. godine ECMA International objavila je šestu glavnu verziju ECMAScripta, koja se službeno naziva ECMAScript 2015, a u početku je nazivana ECMAScript 6 ili ES6. Od tada su ECMAScript standardi u godišnjim ciklusima objavljivanja. JavaScript zaštitni je znak tvrtke Oracle u SAD-u i drugim zemljama. [33]

4.2.1.4. React.js

React je deklarativna, učinkovita i fleksibilna JavaScript biblioteka za izgradnju korisničkih sučelja. Omogućuje sastavljanje složenih korisničkih sučelja iz malih i izoliranih dijelova koda koji se nazivaju komponente [34]. Svaka komponenta ima nekoliko funkcija životnog ciklusa koje se mogu prilagoditi za pokretanje koda u određeno vrijeme u procesu. Te se metode pozivaju sljedećim redoslijedom prilikom nastanka DOM komponente [35]:

- constructor() – koristi se za inicijalizaciju lokalnih varijabla stanja i instanciranje rukovatelja događajima.
- static getDerivedStateFromProps() – koristit se za detekciju promjene svojstava komponenti
- render() – jedina je obavezna funkcija u komponentama. Prikazuje elemente stvorene jsx sintaksom
- componentDidMount () – služi za dohvaćanje podataka s udaljene krajnje točke

React stvara predmemoriju strukture podataka, izračunava razlike i zatim učinkovito ažurira prikaze DOM preglednika. To omogućava programeru pisanje koda kao da se pri svakoj promjeni prikazuje cijela stranica, dok biblioteke React generiraju samo pod-komponente koje se zapravo mijenjaju. Ovaj selektivni prikaz pruža veliko poboljšanje performansi, štedi napor ponovnog izračunavanja CSS stila, izgleda stranice i prikazivanja na cijeloj stranici [36].

4.2.2. Prikaz programskog rješenja klijentske strane po komponentama

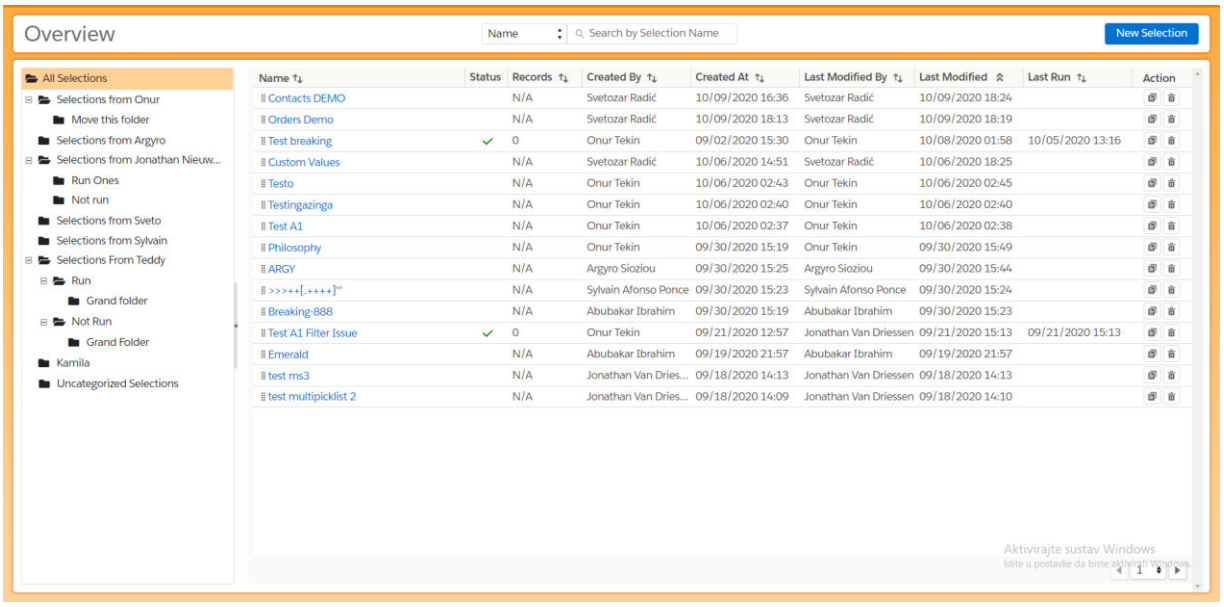
U ovom poglavlju je opisano programsko rješenje po komponentama u kome su dani bitni kodovi za ostvarivanje razvoja okoline.

Kako bismo osigurali lakše snalaženje u kodu klijentska strana organizirana je u sljedeće najvažnije cjeline:

- Api – sadrži funkcije koje služe za komunikaciju s poslužiteljskom stranom
- Components – sadrži dijelove korisničkog sučelja
- Containers – sadrži glavne komponente korisničkog sučelja
- Constants – sadrži pomoćne konstante vrijednosti za ostvarivanje korisničkog sučelja

4.2.2.1. Datoteka Containers

U ovoj datoteci se nalaze glavni dijelovi okoline razvijene u ovom diplomskom radu. Okolina je podijeljena na dva glavna dijela a to su: Overview i Selection. U datoteci Overview nalazi se React komponenta koja predstavlja korisničko sučelja za pregled svih stvorenih SQL upita odnosno „Selekcija“. Prikaz korisničkog sučelja Overview nalazi se na slici 4.42.



Sl. 4.42. Prikaz Overview korisničkog sučelja

Prema slici 4.42, na korisničkom sučelju Overview , korisnik može početi stvarati novu selekciju odnosno novi SQL upit pritiskom na tipku „New Selection“ koja se nalazi u gornjem desnom kutu. Nakon što je nova selekcija stvorena, prikazane su informacije o novonastaloj selekciji. Informacije o selekciji koje Overview pruža su sljedeće: Name (ime selekcije), Status (indikator o uspješnosti izvršenog SQL upita), Records (ukupan broj vrijednosti upisanu u ciljno podatkovno proširenje), Created By (ime i prezime osobe koja je stvorila selekciju), Created At (datum kada je selekcija stvorena), Last Modified By (ime i prezime osobe koja je zadnja uređivala selekciju), Last Modified (datum kada je selekcija zadnji put uređivana), Last Run (datum kada je selekcija zadnji put bila pokrenuta). Osim navedenih informaciji o selekciji korisničko sučelje Overview pruža mogućnost brisanja, uređivanja i kopiranja selekcije, te mogućnost organizacije selekcija po datotekama. Također je implementirana paginacija te mogućnost pretraživanja selekcija po imenu selekcije, osobe koja je stvorila selekciju odnosno osobe koja je zadnji put uređivala selekciju. Korisničko sučelje Overview sadrži komponente Navbar, Folders i Selection list. Komponenta Navbar predstavlja navigacijski dio korisničkog sučelja Overview. Funkcionalnosti koje pruža komponenta Navbar su: pretraživanje selekcija po imenu selekcije, imenu osobe koja je stvorila selekciju te imenu osobe koja je zadnji put uredila selekciju. Funkcija za pretraživanje selekcija nalazi se na slici 4.43.

```
173  /**
174   * Search field handler for selections. It updates the state on every keystroke.
175   * If search field is empty, take user to page one on pagination button one
176   * @param {object} e - e.target
177   * @param {bool} isSelectValue - determines if we are searching by name or by other criteria
178   */
179  handleSearchField = (e, isSelectValue) => {
180    const { selections } = this.state;
181
182    if (isSelectValue) {
183      this.setState({ searchCriteriaValue: e.target.value });
184    } else {
185      this.setState({ searchField: e.target.value }, () => {
186        // After search if we go on page 2 and then delete search input then it comes back on page 1
187        this.setState({ paginationIndex: 1 });
188        this.paginateTheSelections(selections, 1);
189      });
190    }
191  };
```

Sl. 4.43. Prikaz funkcije za pretraživanje selekcije

Prema slici 4.43 vidljivo je kako funkcija za pretraživanje selekcija prima kriterij po kojemu se selekcije pretražuju te postavlja varijablu stanja searchCriteriaValue na trenutnu vrijednost pretraživanja. Nakon što funkcija postavi varijablu stanja na trenutnu vrijednost pretraživanja,

u metodi render(), koristi se varijabla stanja searchCriteriaValue prilikom filtriranja selekcije (Slika 4.44).

```
1021        // Set selections based on different actions
1022        let filteredSelections;
1023        // Input search
1024        if (searchField || searchCriteriaValue) {
1025          // Selections filterd by search input and filter criteria
1026          filteredSelections = [...selections].filter(
1027            // Set length for updatedBy if undefined
1028            (selection) => {
1029              if (selection.updatedBy === undefined) {
1030                // eslint-disable-next-line no-param-reassign
1031                selection.updatedBy = '';
1032              }
1033              if (searchCriteriaValue.length > 0) {
1034                return selection[searchCriteria]
1035                  .toLowerCase()
1036                  .includes(searchCriteriaValue.toLowerCase());
1037              }
1038              return selection[searchCriteria]
1039                .toLowerCase()
1040                .includes(searchField.toLowerCase());
1041            },
1042          );
```

Sl. 4.44. Prikaz filtriranja selekcije

Komponenta Folders korisničkog sučelja Overview služi za organizaciju selekcija po datotekama. Datoteke su organizirane u dvije kategorije: All Selections i Uncategorized Selections. Desnim klikom na datoteku All Selections omogućeno je stvaranje novih datoteka. Osim stvaranja novih datoteka stvorene su funkcionalnosti za brisanje i preimenovanje postojećih datoteka. Funkcija za stvaranje nove datoteke prikazana je na slici 4.45. Selekcija se metodom „*drag and drop*“ smjesti u jednu od ponuđenih datoteka.

```
createNewFolder = async () => {
  const {
    folders, refreshFolders, handleSetAppState, folderId,
  } = this.props;

  let nameExists = true;
  let result;

  // Needed to avoid creating a function each iteration within the while loop
  const findFunction = el => el.name === result.value;

  // If folder name exists show error
  while (nameExists) {
    // eslint-disable-next-line no-await-in-loop
    result = await swal.fire({
      title: 'Folder Name',
      input: 'text',
      showCancelButton: true,
      confirmButtonText: 'Save',
      footer: '<div></div>',
      buttonsStyling: false,
      animation: false,
      inputAttributes: {
        maxlength: 150,
      },
      inputValidator: (value) => {
        if (!value) {
          return 'Folder name cannot be empty.';
        }
        return null;
      },
    });

    const folder = folders.find(findFunction);

    if (folder) {
      // eslint-disable-next-line no-await-in-loop
      await this.showErrorFolderMessage();
    } else {
      nameExists = false;
    }
  }

  // If name is unique create new folder
  if (result.value) {
    try {
      const folder = folders.find(el => el.name === result.value);
      if (folder) {
        await this.showErrorFolderMessage();
        return;
      }

      // Create new folder
      await SelectionFoldersApi.createSelectionFolder(this.axiosCancelToken.token, result.value, folderId);

      const foundFolder = folders.filter(el => el._id === folderId);

      if (foundFolder.length !== 0) {
        handleSetAppState({ folderId: '' });
      }

      refreshFolders();
    } catch (err) {
      this.setState({ error: err });
    }
  } else {
    handleSetAppState({ folderId: '' });
  }
}
```

Sl. 4.45. Prikaz funkcije za stvaranje nove datoteke

Prema slici 4.45 funkcija za stvaranje nove datoteke provjerava postoji li već datoteka s unesenim imenom. Ukoliko postoji pojavljuje se prozor upozorenja s porukom da datoteka s navedenim imenom već postoji. Nakon što je korisnik unio jedinstveno ime poziva se API za stvaranje nove datoteke koji sprema novonastalu datoteku u bazu podataka. Na kraju je potrebno osvježiti listu datoteka kako bi se novonastala datoteka prikazala na korisničkom sučelju. Komponenta Selection list korisničkog sučelja Overview služi za prikaz stvorenih selekcija te informacija vezanih za njih. Ova komponenta omogućuje uređivanje, brisanje i kopiranje postojećih selekcija te metodu „*drag and drop*“ za organizaciju selekcija u datoteke. Selekcije se dohvaćaju pomoću funkcije prikazane na slici 4.46.

```
964   /**
965    * Refresh and set the selections state on the front page
966    * @param {string} filterFolderId - id of the current folder
967    */
968   refreshSelections = async (filterFolderId) => {
969     const { handleSetAppState } = this.props;
970     const retrievedSelections = await SelectionsAPI.getSelections(this.axiosCancelToken.token) || [];
971
972     this.setState({
973       selections: retrievedSelections,
974     });
975     handleSetAppState({ selections: retrievedSelections });
976     const { selections } = this.state;
977     // Refresh the number of pagination buttons
978     const filteredSelectionsByFolderId = this.handleFilterFolderSelections(selections, filterFolderId);
979     this.calculatePageItemsAndPageCount(filteredSelectionsByFolderId, true);
980   };
```

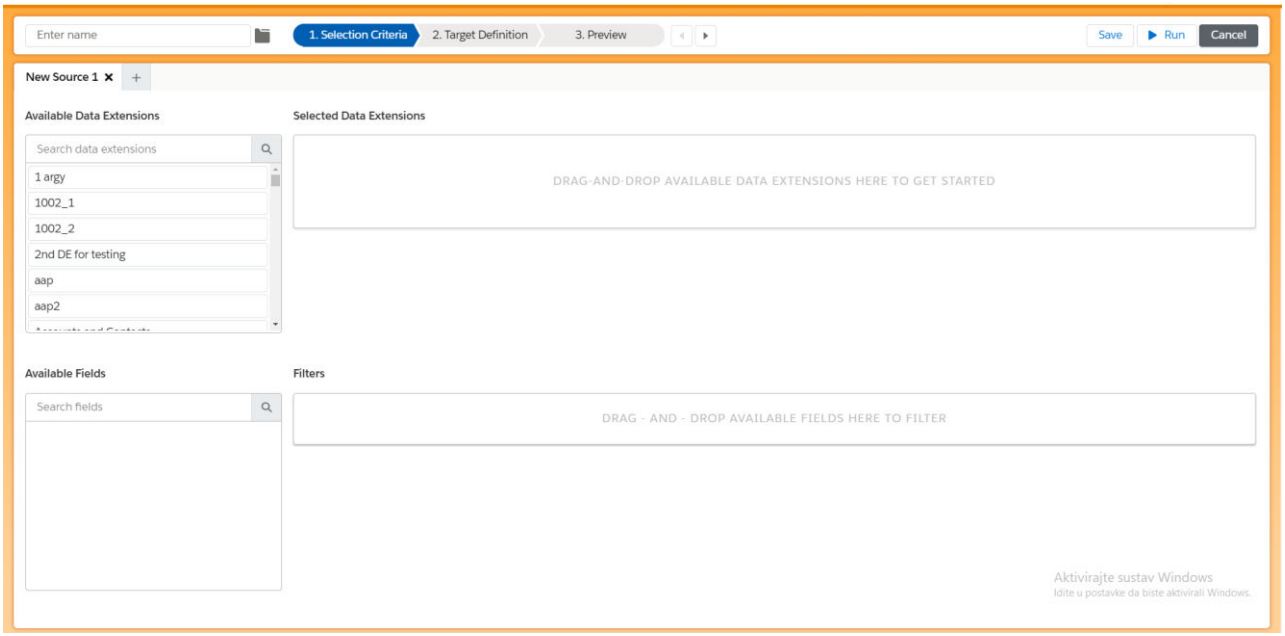
Sl. 4.46. Prikaz funkcije za dohvaćanje selekcija

Prema slici 4.46, funkcija za dohvaćanje selekcija poziva API funkciju getSelections() koja vraća selekcije iz baze podataka te dobivene selekcije sprema u varijablu stanja „selections“. Nakon što se selekcija dohvatila iz baze podataka pomoću funkcija handleFilterFolderSelections() i calculatePageItemsAndPageCount() dobivene se selekcije smještaju u pripadajuće datoteke te se računa broj potrebnih stranica na kojima će se dobivene selekcije prikazati. Računanje broja stranica potrebno je iz razloga što je maksimalan broj selekcija koje se mogu prikazati na jednoj stranici ograničen na 20 selekcija po stranici. Ukoliko se želi obrisati selekcija poziva se funkcija prikazana na slici 4.47.

```
/**
 * Delete selected selection by its id
 * Ask user if he wants to delete the selection (Swal pop up)
 * @param {string} selectionId - selection's id
 */
handleRemoveSelection = async (selectionId) => {
  const result = await swal.fire({
    title: 'Delete Selection',
    // eslint-disable-next-line max-len
    html: '<p class="width_swal">Are you sure you want to delete this selection? <br /> You won\'t be able to revert
      this!</p>',
    type: 'warning',
    showCancelButton: true,
    confirmButtonColor: 'red',
    confirmButtonText: 'Yes, delete it',
    footer: '<div></div>',
    buttonsStyling: false,
  });

  const { numberOfItemsPerPage } = this.state;

  if (result.value) {
    try {
      await SelectionsAPI.deleteSelection(
        selectionId,
        this.axiosCancelToken.token,
      );
      await this.refreshSelections();
      // After deleting last item from last page, set pagination index to the last available one
      const { pageItems, selections, paginationIndex } = this.state;
      if (
        selections.length / numberOfItemsPerPage <= pageItems.length &&
        paginationIndex > pageItems.length
      ) {
        this.setState({
          paginationIndex: pageItems.length,
        });
      }
    } catch (error) {
      this.setState({ error });
    }
  }
};
```

Sl. 4.47. Prikaz funkcije za brisanje selekcije

Prema slici 4.47, funkcija za brisanje selekcije pruža korisniku poruku upozorenja kako jednom obrisana selekcija ne može biti vraćena. Ukoliko korisnik potvrdi brisanje selekcije, funkcija poziva API koji obriše odabranu selekciju iz baze podataka. Nakon toga se osvježe trenutne selekcije kako obrisana selekcije ne bi bila više prikazana korisniku. Pritiskom na ime selekcije otvara se odabrana selekcija. U datoteci Selection nalazi se komponenta koja predstavlja korisničko sučelja za stvaranje SQL upita odnosno selekcije kroz tri glavna koraka. Prikaz korisničkog sučelja Selection nalazi se na slici 4.48.



Sl. 4.48. Prikaz korisničkog sučelja za stvaranje selekcije

Prema 4.48 vidljivo je kako korisničko sučelje Selection sadrži tri koraka pri stvaranju nove selekcije koji su prikazani u gornjem dijelu sučelja: Selection Criteria, Target Definition i Preview. Također, gornji dio sučelja sadrži polje za unos imena selekcije te tipke za spremanje selekcije, pokretanje selekcije i otkazivanje selekcije. Prvi korak za stvaranje SQL upita odnosno selekcije je definiranje izvorišnih podatkovnih proširenja, odnose između njih te definiranje filtera. Kako bi bilo moguće definiranje izvorišnih podatkovnih proširenja korisničko sučelje Selection sadrži listu svih dostupnih podatkovnih proširenja koja se nalaze u sekciji Available Data Extension. Dostupna proširenja se dohvaćaju u funkciji componentDidMount() koja se poziva prilikom prikazivanja komponente Selection. Nakon što se podatkovna proširenja dohvate pozivom API-ja sortiraju se abecednim redom radi lakše čitljivosti. Kod zadužen za dohvaćanje i sortiranje podatkovnih proširenja prikazan je na slici 4.49.

```
194        const dataExtensions = await DataExtensionsAPI.getDataExtensions(this.axiosCancelToken.token,
195          Constants.DATAEXTENSION__FILTER_MODE__AVAILABLE);
196        this.formatAvailableDEs(dataExtensions);
197        this.setState({
198          dataExtensions,
199          isDataExtensionRequestDone: true,
200        });
```

Sl. 4.49. Prikaz koda za dohvaćanje i sortiranje podatkovnih proširenja

Prema slici 4.49 vidljivo je kako se poslije poziva funkcije getDataExtension(), kojom se poziva API za dohvaćanje podatkovnih proširenja iz Salesforce Marketing Clouda, poziva

funkcija za sortiranje dobivenih podatkovnih proširenja. Funkcija za sortiranje podatkovnih proširenja prima polje podatkovnih proširenja te ih sortira pomoću funkcije formatAvailableDEs(). Funkcija za sortiranje, sortArrayOfObjects(), poziva se unutar funkcije formatAvailableDEs() te je prikazana na slici 4.50.

```
52    /**
53     * Sort array of object by 'sortBy'
54     * @param {array} array - The array to sort
55     * @param {string} sortBy - The key of the array to sort by
56     * @returns {array} The sorted array
57     */
58    sortArrayOfObjects: (array, sortBy) => {
59      array.sort((a, b) => {
60        if (a[sortBy].toUpperCase() < b[sortBy].toUpperCase()) return -1;
61        if (a[sortBy].toUpperCase() > b[sortBy].toUpperCase()) return 1;
62        return 0;
63      });
64    },
```

Sl. 4.50. Prikaz funkcije za sortiranje polja podatkovnih proširenja

Nakon što su podatkovna proširenja dohvaćena i sortirana korisnik pomoću metode „*drag and drop*“ može premjestit željeno podatkovno proširenje u sekciju Selected Data Extensions. Najprije je stvorena sekcija Selected Data Extensions pomoću jsx sintakse prikazane na slici 4.51.

```jsx
<div className="selected-extension">
  <h4 className="section-title">Selected Data Extensions</h4>
  <div
    id="selected-collections"
    className="selected-extension_wrapper"
    onDragOver={e => e.preventDefault()}
    onDrop={(e) => {
      // Needed to prevent a redirect
      e.preventDefault();
      handleSetSelectionState({ DEBorderMouseOver: false });
      noBorderOnDrop(e);
      return dropOnSelectedCollections(e, handleDropToSelectedDE);
    }}
    onDragEnter={e => borderOnEnter(e)}
    onDragLeave={e => borderOnLeave(e)}
    style={{ pointerEvents: filterBorderMouseOver ? 'none' : '' }}
  >
    {selectedDataExtensions.length <= 0 ?
      (
        <p
          id="drag-and-drop-paragraph"
          style={{
            pointerEvents: DEBorderMouseOver ? 'none' : '',
            border: DEBorderMouseOver ? 'var(--dashedBorder)' : 'var(--transparentBorder)',
          }}
        >
          Drag-and-drop available data extensions here to get started
        </p>
      ) :
      (
        selectedDataExtensions.map((dataExtension, i) => dataExtension.dragged && i < 1 ?
          (
            <AvailableDECard
              key={dataExtension.id}
              // eslint-disable-next-line react/jsx-props-no-spreading
              {...dataExtension}
              handleDeleteSelectedDE={handleDeleteSelectedDE}
              handleSameAliasErrorPopup={handleSameAliasErrorPopup}
              matchedFields={matchedFields}
              handleSetSelectionState={handleSetSelectionState}
              filtersRef={filtersRef}
              selectedDataExtensions={selectedDataExtensions}
              handleAliasPopUp={handleAliasPopUp}
              relations={relations}
              DEBorderMouseOver={DEBorderMouseOver}
              filterBorderMouseOver={filterBorderMouseOver}
            />
          ) :
          null)
      )}
  </div>
</div>
</>
```

Sl. 4.51. Prikaz jsx sintakse za stvaranje sekcije Selected Data Extensions

Prema slici 4.51 vidljivo je kako se prilikom akcije „*on drop*“ na div elementu poziva funkcija dropOnSelectedCollections() koja implementira metodu „*drag and drop*“ (Slika 4.52).

```
/**
 * Checks if there is data extension in the selected data extensions section
 * Then calls handleDrop func. to updates related data and sets it
 * @param {object} event - event
 * @param {object} handleDrop - callback
 */
const dropOnSelectedCollections = async (event, handleDrop) => {
  event.preventDefault();
  const customerKey = event.dataTransfer.getData('data-collection-customerkey');
  const name = event.dataTransfer.getData('data-collection-name');

  // If selected data extension exists already (on second drop)
  if (selectedDataExtensions.length > 0) {
    /*
     * Force user to drop the next DE on an existing DE in order to create a relation
     * Avoid any other location
     */
    if (
      (event.target.nodeName.toLowerCase() !== 'div' &&
    event.target.nodeName.toLowerCase() !== 'span' &&
    event.target.nodeName.toLowerCase() !== 'button' &&
    event.target.nodeName.toLowerCase() !== 'i' &&
    event.target.nodeName.toLowerCase() !== 'svg' &&
    event.target.nodeName.toLowerCase() !== 'use') ||
  event.target.id === 'selected-collections'
    ) {
      // Display error
      swal.fire({
        type: 'error',
        title: '<div class="error-title">Oops...</div>',
        html: name ?
          /* eslint-disable max-len */
          '<p class="width_swal">Please drop your data extension on another selected data extension to define a
            relationship</p>' :
          '<p class="width_swal">You can only drop data extensions here.</p>',
        /* eslint-enable max-len */
        footer: '<div></div>',
        buttonsStyling: false,
        animation: false,
      });
      return null;
    }
    // Search for the   id   of the parent to which newly added DE will have relation with
    let toDataExtensionNodeId;

    // If user drops on nodeName: button/span/or relation sentence - get parentNode's id
    if (
      event.target.nodeName.toLowerCase() === 'span' ||
      event.target.nodeName.toLowerCase() === 'button' ||
      (event.target.nodeName.toLowerCase() === 'div' &&
      event.target.className !== 'available-data-extension dragged-collection nohover')
    ) {
      toDataExtensionNodeId = event.target.parentNode.id;
    // If user drops on nodeName: 'i' - get its parent's parent id (because i's parent is a button)
    } else if (
      (event.target.nodeName.toLowerCase() === 'i' &&
      event.target.parentNode.parentNode.nodeName.toLowerCase() === 'div') ||
      event.target.nodeName.toLowerCase() === 'svg'
    ) {
      toDataExtensionNodeId = event.target.parentNode.parentNode.parentNode.id;
    } else if (event.target.nodeName.toLowerCase() === 'use') {
      toDataExtensionNodeId = event.target.parentNode.parentNode.parentNode.parentNode.id;
    } else {
      toDataExtensionNodeId = event.target.id;
    }

    handleDrop(customerKey, name, toDataExtensionNodeId);
  } else {
    // On initial drop
    handleDrop(customerKey, name);
  }
  return null;
};
```

Sl. 4.52. Prikaz funkcije dropOnSelectedCollections

Prema slici 4.52 vidljivo je kako se u navedenoj funkciji provjerava postoji li već podatkovno proširenje u sekciji Selected Data Extensions te ukoliko postoji provjerava se da li je podatkovno proširenje spušteno na odgovarajuće HTML elemente (npr. oznaka span). Ukoliko je podatkovno proširenje neispravno spušteno poziva se funkcija swal.fire() koja prikazuje upozoravajuću poruku korisniku. No, ako je podatkovno proširenje spušteno na odgovarajuće mjesto poziva se funkcija handleDrop() koja je zadužena za postavljanje varijable stanja „selectedDataExtensions“ te stvaranja veze između podatkovnih proširenja. Prilikom stvaranja veza između podatkovnih proširenja koriste se funkcije handleRelationModalFields() i handleRelationModalSave(). Funkcija handleRelationModalFields() određuje koje je polje iz prvog podatkovnog proširenja pridruženo polju drugog podatkovnog proširenja (Slika 4.53).

```
142   /**
143    * Adjust relation fields in relational modal
144    * Avoid mutation
145    * @param {object} event - event
146    * @param {number} toFromIndex - from index
147    */
148   handleRelationalModalFields = (event, toFromIndex) => {
149     const { modalDataExtensions } = this.state;
150     const { handleSetSelectionState } = this.props;
151
152     // Shallow copies to prevent reference problem
153     const modalDataExtensionsCopy = { ...modalDataExtensions };
154
155     /**
156      * 0 <- options on left, options on right -> 1
157      * modalDataExtensionsCopy.formCollection.fromField <- current choice
158      * event.target.value <- new choice
159      */
160     if (toFromIndex === 0) {
161       const fromCollectionCopy = { ...modalDataExtensionsCopy.fromCollection };
162       fromCollectionCopy.fromField = event.target ?
163         event.target.value :
164         event;
165
166       modalDataExtensionsCopy.fromCollection = fromCollectionCopy;
167     } else if (toFromIndex === 1) {
168       const toCollectionCopy = { ...modalDataExtensionsCopy.toCollection };
169       toCollectionCopy.toField = event.target ?
170         event.target.value :
171         event;
172
173       modalDataExtensionsCopy.toCollection = toCollectionCopy;
174     }
175
176     // Update state in Selection with new modal data extension
177     handleSetSelectionState({ modalDataExtensions: modalDataExtensionsCopy });
178   };
```

Sl. 4.53. Prikaz funkcije handleRelationModalFields()

Funkcija handleRelationModalSave() zadužena je za postavljanje varijabli stanja potrebnih za stvaranje odnosa između podatkovnih proširenja (Slika 4.54).

```
110   /**
111    * Save the new or changed relationship
112    */
113   handleRelationalModalSave = () => {
114     const { modalDataExtensions } = this.state;
115     const { handleSetSelectionState, relations } = this.props;
116
117     const relation = relations ?
118       relations.filter(r => r.relationalModalId === modalDataExtensions.relationalModalId) :
119       null;
120     if (relation && relation.length > 0) {
121       for (let i = 0; i < relations.length; i += 1) {
122         if (relations[i].relationalModalId === modalDataExtensions.relationalModalId) {
123           relations[i] = modalDataExtensions;
124           break;
125         }
126       }
127       this.setState({ showRelationalModal: false });
128       handleSetSelectionState({ relations, modalDataExtensions: {} });
129     } else {
130       /*
131        * Update state in Selection with new relation
132        * Close modal
133        */
134       this.setState({ showRelationalModal: false });
135       handleSetSelectionState(prevState => ({
136         relations: [...prevState.relations, modalDataExtensions],
137         modalDataExtensions: {},
138       }));
139     }
140   };
```

Sl. 4.54. Prikaz funkcije handleRelationModalSave()

Nakon što su odabrana podatkovna proširenja i definirane veze između njih moguće je stvoriti filtre kako bi se ograničili vrijednosti izvorišnih podatkovnih proširenja. Filtri se stvaraju tako što se željeno polje prije odabranog podatkovnog proširenja metodom „*drag and drop*“ povuče u sekciju za filtre. Nakon što se odabrano polje povuče u sekciju za filtre potrebno je definirati kriterij filtra i vrijednost s kojom će se vrijednost iz povučenog polja uspoređivati. Ovisno o tipu odabranog polja prikazuju se pojedini kriteriji za filtriranje (Slika 4.55).

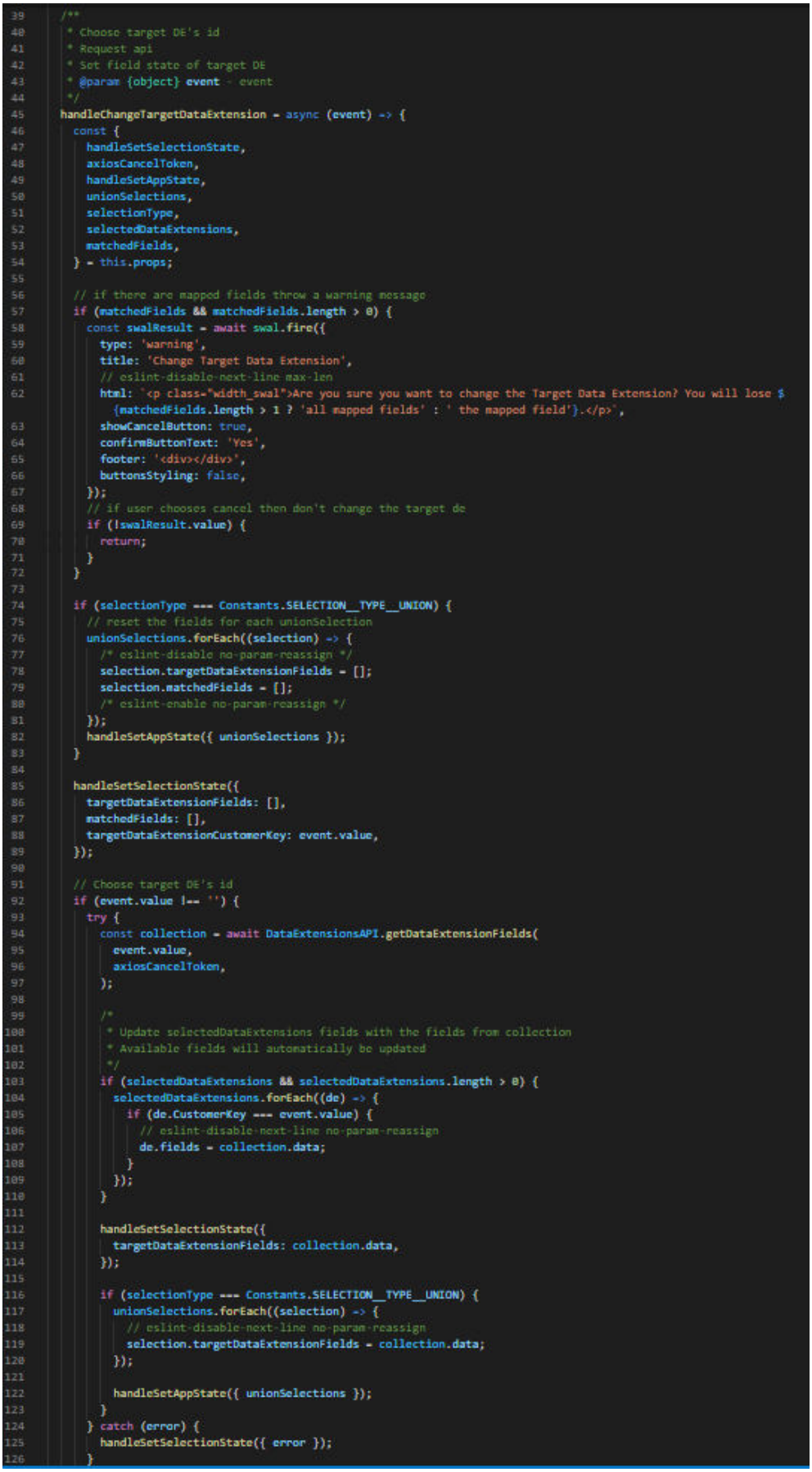
Sl. 4.55. Prikaz kriterija za filtriranje

Nakon što se odabere željeni kriterij za filtriranje poziva se funkcija handleUpdateFilterLineCriteria() koja sprema vrijednost odabranog kriterija u varijablu stanja „criteria“. Osim regularnih filtra okolina pruža mogućnost stvaranja pod-upita. Funkcija za spremanje pod-upita prikazan je na slici 4.56.

```
/**
 * Update values on filterline
 * @param {string} id - Id of current filter
 * @param {object} subQueryFilters - Filters of sub query
 * @param {object[]} filtersBranch - Filters
 */
const handleUpdateSubQuery = (id, subQueryFilters, filtersBranch) => {
  const { filters } = selectedFilters;
  /*
   * This is the filters array we need to find a matching element on:
   * works on the filters state unless a specific branch is passed
   */
  const filtersArray = filtersBranch || filters;
  // Loop through filtersArray
  for (let f = 0; f < filtersArray.length; f += 1) {
    // If the array element contains a filters array, execute this function on this branch
    if (filtersArray[f].filters) {
      handleUpdateSubQuery(id, subQueryFilters, filtersArray[f].filters);
    } else {
      if (filtersArray[f].id === id) {
        // define a subQuery object if none exists yet
        if (!filtersArray[f].subQuery) {
          filtersArray[f].subQuery = {};
        }
        // set the filters of the subQuery to the filters shown in this subquerymodal
        filtersArray[f].subQuery.filters = subQueryFilters;
      }
    }
  }
  handleSetSelectedFiltersState(
    {
      filters: filtersArray,
    },
  );
};
```

Sl. 4.56. Prikaz funkcije za spremanje pod-upita

Prema slici 4.56 vidljivo je kako se prilikom stvaranja pod-upita mogu stvarati novi filtri koji se koriste kao filtri pod-upita prilikom stvaranja konačnog SQL upita. Nakon što je korisnik definirao filtre prelazi se na sljedeći korak gdje se definira ciljno podatkovno proširenje. Drugi je korak, prilikom stvaranja SQL upita, definiranje ciljnog podatkovnog proširenja u koje će biti upisane krajnje vrijednosti SQL upita. U ovom koraku postoje dva načina definiranja ciljnog podatkovnog proširenja. Prvi način je odabir postojećeg ciljnog proširenja ponuđenih u padajućem izborniku. Prilikom odabira ciljnog podatkovnog proširenja poziva se funkcija handleChangeTargetDataExtension() koja poziva API za dohvaćanje polja odabranog podatkovnog proširenja (Slika 4.57).



```
  /**
   * Choose target DE's id
   * Request api
   * Set field state of target DE
   * @param {object} event - event
   */
  handleChangeTargetDataExtension = async (event) => {
    const {
      handleSetSelectionState,
      axiosCancelToken,
      handleSetAppState,
      unionSelections,
      selectionType,
      selectedDataExtensions,
      matchedFields,
    } = this.props;

    // if there are mapped fields throw a warning message
    if (matchedFields && matchedFields.length > 0) {
      const swalResult = await swal.fire({
        type: 'warning',
        title: 'Change Target Data Extension',
        // eslint-disable-next-line max-len
        html: '<p class="width_swal">Are you sure you want to change the Target Data Extension? You will lose $
          {matchedFields.length > 1 ? 'all mapped fields' : ' the mapped field'}.</p>',
        showCancelButton: true,
        confirmButtonText: 'Yes',
        footer: '<div></div>',
        buttonsStyling: false,
      });
      // if user chooses cancel then don't change the target de
      if (!swalResult.value) {
        return;
      }
    }

    if (selectionType === Constants.SELECTION__TYPE__UNION) {
      // reset the fields for each unionSelection
      unionSelections.forEach((selection) => {
        /* eslint-disable no-param-reassign */
        selection.targetDataExtensionFields = [];
        selection.matchedFields = [];
        /* eslint-enable no-param-reassign */
      });
      handleSetAppState({ unionSelections });
    }

    handleSetSelectionState({
      targetDataExtensionFields: [],
      matchedFields: [],
      targetDataExtensionCustomerKey: event.value,
    });

    // Choose target DE's id
    if (event.value !== '') {
      try {
        const collection = await DataExtensionsAPI.getDataExtensionFields(
          event.value,
          axiosCancelToken,
        );

        /*
         * Update selectedDataExtensions fields with the fields from collection
         * Available fields will automatically be updated
         */
        if (selectedDataExtensions && selectedDataExtensions.length > 0) {
          selectedDataExtensions.forEach((de) => {
            if (de.CustomerKey === event.value) {
              // eslint-disable-next-line no-param-reassign
              de.fields = collection.data;
            }
          });
        }

        handleSetSelectionState({
          targetDataExtensionFields: collection.data,
        });

        if (selectionType === Constants.SELECTION__TYPE__UNION) {
          unionSelections.forEach((selection) => {
            // eslint-disable-next-line no-param-reassign
            selection.targetDataExtensionFields = collection.data;
          });

          handleSetAppState({ unionSelections });
        }
      } catch (error) {
        handleSetSelectionState({ error });
      }
```

Sl. 4.57. Prikaz funkcije za odabir ciljnog podatkovnog proširenja

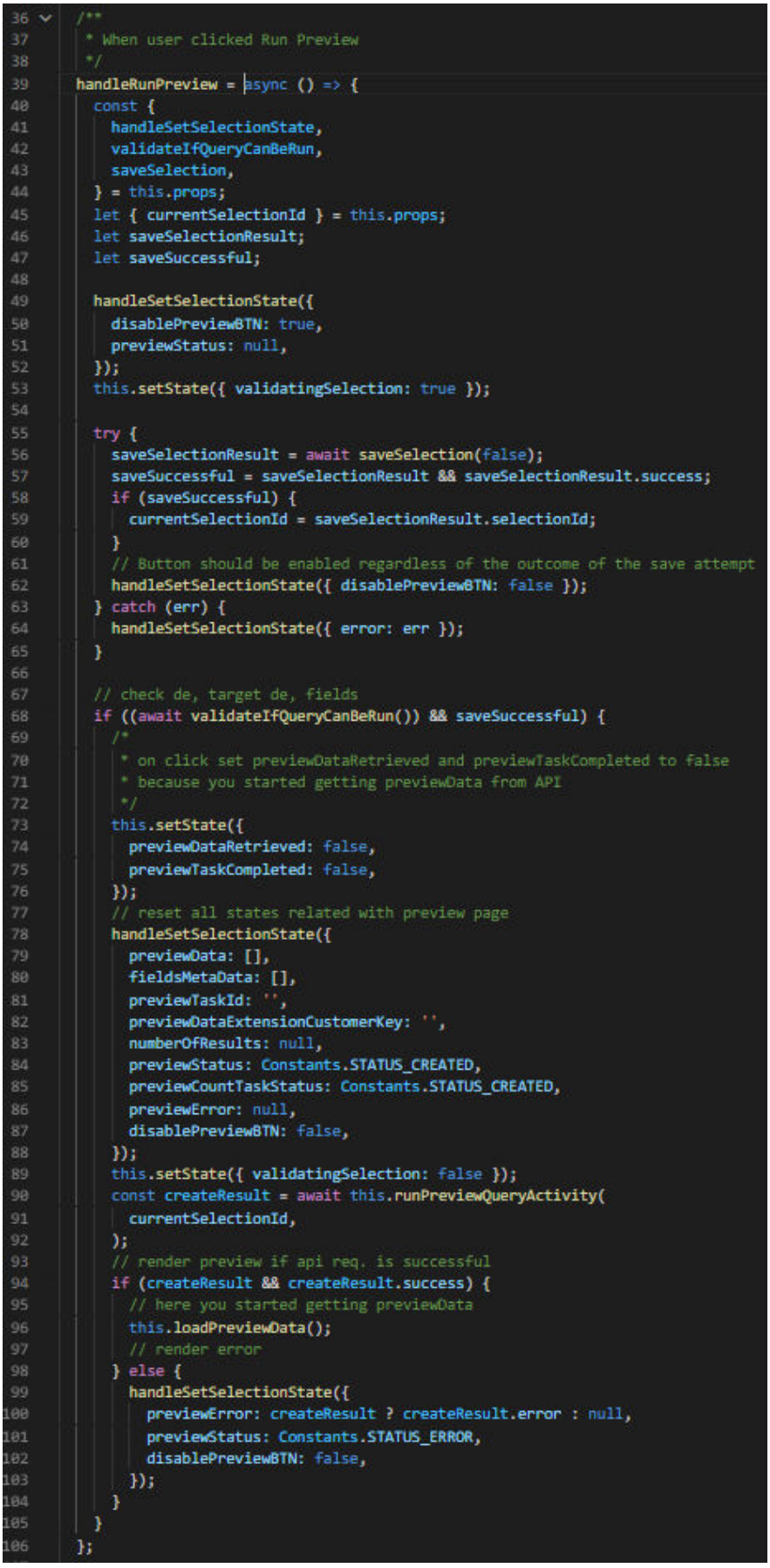
Nakon što se odabere ciljno podatkovno proširenje potrebno je pridružiti polja iz izvorišnog podatkovnog proširenja poljima iz ciljnog podatkovnog proširenja koristeći metodu „*drag and drop*“. Drugi način definiranja ciljnog podatkovnog proširenja je stvaranje novog podatkovnog proširenja. Prilikom stvaranja novog podatkovnog proširenja potrebno je definirati polja koje će novonastalo podatkovno proširenje sadržavati. Polja se definiraju tako što se metodom „*drag and drop*“ povlače polja iz izvorišnog podatkovnog proširenja koja se prosljeđuju funkciji handleCreateTargetDE() za stvaranje novog podatkovnog proširenja. Funkcija handleCreateTargetDE najprije poziva API za stvaranje novog podatkovnog proširenja nakon čega se poziva funkcija za dohvaćanje polja novonastalog podatkovnog proširenja (Slika 4.58).

```
414       createResult = await DataExtensionsAPI.createDataExtension(
415         axiosCancelToken,
416         newTargetDataExtension.folderId,
417         newTargetDataExtension.name,
418         newTargetDataExtension.description,
419         newTargetDataExtensionFields,
420         newTargetDataExtension.dataRetentionPolicy,
421         newTargetDataExtension.relationship,
422       );
423     } catch (error) {
424       createResult = { error: error.response.data.error };
425     }
426
427     // If error
428     if (createResult.error) {
429       // Set newDELoading state
430       this.setState({ newDELoading: false });
431       return swal.fire({
432         type: 'error',
433         title: '<div class="error-title">Error</div>',
434         html: `<p class="width_swal">${createResult.error}</p>`,
435         footer: '<div></div>',
436         buttonsStyling: false,
437         animation: false,
438       });
439     }
440     // If no error
441     if (createResult.newTargetDataExtension && createResult.newTargetDataExtension.CustomerKey) {
442       // Get fields of the newly created targeted DE
443       const newDeFieldsResponse = await DataExtensionsAPI.getDataExtensionFields(
444         createResult.newTargetDataExtension.CustomerKey,
445         axiosCancelToken,
446       );
```

Sl. 4.58. Prikaz pozivanja API funkcija za stvaranje novog podatkovnog proširenja

Nakon što su se obavili prethodno potrebni koraci za stvaranje SQL upita komponenta View pruža mogućnost pokretanja SQL upita pritiskom na tipku „Run Preview“. Pritiskom na navedenu tipku poziva se funkcija handleRunPreview() pomoću koje se prosljeđuju potrebni podaci poslužiteljskoj strani koja stvara i pokreće SQL upit te vraća konačne vrijednosti

klijentskoj strani. Konačne vrijednosti prikazuju se krajnjem korisniku u obliku tablice. Funkcija za pokretanje SQL upita prikazana je na slici 4.59.



```
/**
 * When user clicked Run Preview
 */
handleRunPreview = async () => {
  const {
    handleSetSelectionState,
    validateIfQueryCanBeRun,
    saveSelection,
  } = this.props;
  let { currentSelectionId } = this.props;
  let saveSelectionResult;
  let saveSuccessful;

  handleSetSelectionState({
    disablePreviewBTN: true,
    previewStatus: null,
  });
  this.setState({ validatingSelection: true });

  try {
    saveSelectionResult = await saveSelection(false);
    saveSuccessful = saveSelectionResult && saveSelectionResult.success;
    if (saveSuccessful) {
      currentSelectionId = saveSelectionResult.selectionId;
    }
    // Button should be enabled regardless of the outcome of the save attempt
    handleSetSelectionState({ disablePreviewBTN: false });
  } catch (err) {
    handleSetSelectionState({ error: err });
  }

  // check de, target de, fields
  if ((await validateIfQueryCanBeRun()) && saveSuccessful) {
    /*
     * on click set previewDataRetrieved and previewTaskCompleted to false
     * because you started getting previewData from API
     */
    this.setState({
      previewDataRetrieved: false,
      previewTaskCompleted: false,
    });
    // reset all states related with preview page
    handleSetSelectionState({
      previewData: [],
      fieldsMetaData: [],
      previewTaskId: '',
      previewDataExtensionCustomerKey: '',
      numberOfResults: null,
      previewStatus: Constants.STATUS_CREATED,
      previewCountTaskStatus: Constants.STATUS_CREATED,
      previewError: null,
      disablePreviewBTN: false,
    });
    this.setState({ validatingSelection: false });
    const createResult = await this.runPreviewQueryActivity(
      currentSelectionId,
    );
    // render preview if api req. is successful
    if (createResult && createResult.success) {
      // here you started getting previewData
      this.loadPreviewData();
      // render error
    } else {
      handleSetSelectionState({
        previewError: createResult ? createResult.error : null,
        previewStatus: Constants.STATUS_ERROR,
        disablePreviewBTN: false,
      });
    }
  }
};
```

Sl. 4.59. Prikaz funkcije za pokretanje SQL upita

4.2.2.2. Datoteka Api

U datoteci Api nalaze se funkcije koje implementiraju API pozive kako bi dobili podatke od poslužiteljske strane odnosno kako bi poslali potrebne podatke poslužiteljskoj strani u svrhu stvaranja SQL upita. Za ostvarivanje komunikacije s poslužiteljskom stranom API funkcije koriste biblioteku axios. Datoteka Api sadrži sljedeće API funkcije:

- API funkcije za rad s podatkovnim proširenjima – dohvaćanje podatkovnih proširenja i polja podatkovnih proširenja, stvaranje novih proširenja i uređivanje postojećih proširenja. Prikaz API funkcija za rad s podatkovnim proširenjima nalazi se na slici 4.60.

```
const DataExtensionsAPI = {
  getDataExtensionData: async (customerKey, cancelToken, limit) => {
    const res = await axios.get(
      `${apiUrl}/dataextensions/${customerKey}/data?limit=${limit}`,
      Util.apiGetCallSettings(cancelToken),
    );
    return res.data.data;
  },

  /**
   * Get one specific data extensions
   * @param {string} customerKey - DE customer key
   * @param {object} cancelToken - token axios
   */
  getDataExtensionFields: async (customerKey, cancelToken) => {
    const res = await axios.get(
      `${apiUrl}/dataextensions/${customerKey}/fields`,
      Util.apiGetCallSettings(cancelToken),
    );
    return res.data;
  },

  /**
   * Get all data extensions for org
   * @param {object} cancelToken - token axios
   * @param {string} mode - 1 on Selection Criteria or 2 on Target Definition. Numbers needed for the backend
   */
  getDataExtensions: async (cancelToken, mode) => {
    const res = await axios.get(`${apiUrl}/dataextensions?mode=${mode}`, Util.apiGetCallSettings(cancelToken));
    return res.data.data;
  },

  /**
   * Create new (target) data extension
   * @param {object} cancelToken - token axios
   * @param {number} folderIn - number of the folder DE will be in
   * @param {string} name - name of new DE
   * @param {string} description - description
   * @param {object[]} fields - fields in target definition
   * @param {object} retentionPolicy - data retention policy
   */
  createDataExtension: async (cancelToken, folderId, name, description, fields, retentionPolicy, relationship) => {
    const postData = {
      folderId,
      name,
      description,
      fields,
      retentionPolicy,
      relationship,
    };
    const res = await axios.post(`${apiUrl}/dataextensions/create`, postData, Util.apiPostCallSettings(cancelToken));
    return res.data;
  },

  /**
   * Update Target Data Extension fields
   * @param {object} cancelToken - token axios
   * @param {object[]} fields - fields in target definition
   * @param {String} customerKey - Customer Key of the Target Data Extension we are going to update
   */
  updateTargetDataExtensionFields: async (cancelToken, fields, customerKey) => {
    const postData = {
      fields,
      customerKey,
    };
    const res = await axios.post(`${apiUrl}/dataextensions/addField`, postData, Util.apiPostCallSettings(cancelToken));
    return res.data;
  },
};
```

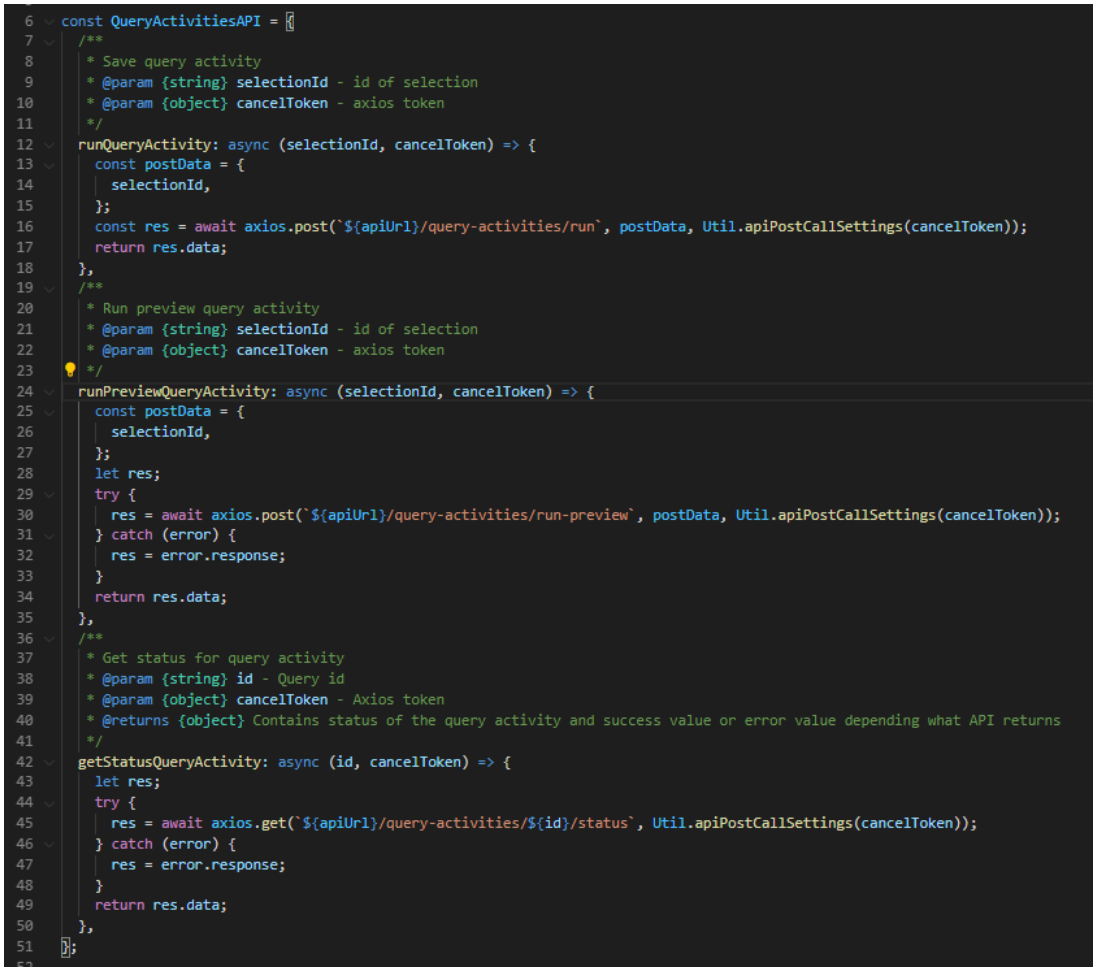
Sl. 4.60. Prikaz API funkcija za rad s podatkovnim proširenjima

- API funkcije za datotečnu organizaciju selekcija – dohvaćanje postojećih datoteka, stvaranje novih datoteka, uređivanje i brisanje datoteka. Prikaz API funkcija za rad s datotekama nalazi se na slici 4.61.

```
const SelectionFoldersAPI = {
  /**
   * Get all selection folders
   * @param {object} cancelToken - Axios token
   * @returns {array} Array of selection folders
   */
  getSelectionFolders: async (cancelToken) => {
    const res = await axios.get(`${apiUrl}/selection-folders/`, Util.apiGetCallSettings(cancelToken));
    return res.data;
  },
  /**
   * Create new selection folder
   * @param {object} cancelToken - Axios token
   * @param {object} name - Name of selectionfolder
   * @param {object} parentFolderId - ParentFolderId of selectionfolder
   * @returns {object} Object of newly created folder
   */
  createSelectionFolder: async (cancelToken, name, parentFolderId) => {
    const res = await axios.post(
      `${apiUrl}/selection-folders/`,
      { name, parentFolderId },
      Util.apiPostCallSettings(cancelToken),
    );
    return res.data;
  },
  /**
   * Update selection folder
   * @param {object} cancelToken - Axios token
   * @param {string} folderId - Id of folder
   * @param {object} name - Name of selectionfolder
   * @param {object} parentFolderId - ParentFolderId of selectionfolder
   * @returns {object} Object of updated folder
   */
  updateSelectionFolder: async (cancelToken, folderId, name, parentFolderId) => {
    const res = await axios.put(
      `${apiUrl}/selection-folders/${folderId}`,
      { name, parentFolderId },
      Util.apiPostCallSettings(cancelToken),
    );
    return res.data;
  },
  /**
   * Delete selection folder
   * @param {object} cancelToken - Axios token
   * @param {string} folderId - Id of folder
   * @returns {object} Object of deleted folder
   */
  deleteSelectionFolder: async (cancelToken, folderId) => {
    const res = await axios.post(
      `${apiUrl}/selection-folders/${folderId}/delete`,
      {},
      Util.apiPostCallSettings(cancelToken),
    );
    return res.data;
  },
};
```

Sl. 4.61. Prikaz API funkcija za rad s datotekama

- API funkcije za rad s SQL aktivnostima – pokretanje SQL aktivnosti, pokretanje pogleda SQL aktivnosti i dobivanje trenutnog status pokrenute SQL aktivnosti. Prikaz API funkcija za rad s SQL aktivnostima nalazi se na slici 4.62.



```
const QueryActivitiesAPI = {
  /**
   * Save query activity
   * @param {string} selectionId - id of selection
   * @param {object} cancelToken - axios token
   */
  runQueryActivity: async (selectionId, cancelToken) => {
    const postData = {
      selectionId,
    };
    const res = await axios.post(`${apiUrl}/query-activities/run`, postData, Util.apiPostCallSettings(cancelToken));
    return res.data;
  },
  /**
   * Run preview query activity
   * @param {string} selectionId - id of selection
   * @param {object} cancelToken - axios token
   */
  runPreviewQueryActivity: async (selectionId, cancelToken) => {
    const postData = {
      selectionId,
    };
    let res;
    try {
      res = await axios.post(`${apiUrl}/query-activities/run-preview`, postData, Util.apiPostCallSettings(cancelToken));
    } catch (error) {
      res = error.response;
    }
    return res.data;
  },
  /**
   * Get status for query activity
   * @param {string} id - Query id
   * @param {object} cancelToken - Axios token
   * @returns {object} Contains status of the query activity and success value or error value depending what API returns
   */
  getStatusQueryActivity: async (id, cancelToken) => {
    let res;
    try {
      res = await axios.get(`${apiUrl}/query-activities/${id}/status`, Util.apiPostCallSettings(cancelToken));
    } catch (error) {
      res = error.response;
    }
    return res.data;
  },
};
```

Sl. 4.62. Prikaz API funkcija za rad s SQL aktivnostima

- API funkcije za rad sa selekcijama – dohvaćanje selekcija, kopiranje selekcija, uređivanje i brisanje postojećih. Prikaz API funkcija za rad sa selekcijama nalazi se na slici 4.63.

```
  /**
   * Get all Selections
   * @param {object} cancelToken - token axios
   */
  getSelections: async (cancelToken) => {
    const res = await axios.get(`${apiUrl}/selections`, Util.apiGetCallSettings(cancelToken));
    return res.data.selections;
  },

  /**
   * Creates copy of the selection
   * @param {string} selectionId - id of selection
   * @param {object} cancelToken - axios token
   */
  copySelection: async (selectionId, cancelToken) => {
    const res = await axios.post(
      `${apiUrl}/selections/${selectionId}/copy`,
      {},
      Util.apiPostCallSettings(cancelToken),
    );
    return res.data;
  },

  /**
   * deletes selection
   * @param {string} selectionId - id of selection
   * @param {object} cancelToken - axios token
   */
  deleteSelection: async (selectionId, cancelToken) => {
    const res = await axios.post(
      `${apiUrl}/selections/${selectionId}/delete`,
      {},
      Util.apiPostCallSettings(cancelToken),
    );
    return res.data;
  },

  /**
   * update selection
   * @param {string} selectionId - id of selection
   * @param {string} folderId - id of folder
   * @param {object} cancelToken - axios token
   */
  updateSelection: async (selectionId, folderId, cancelToken) => {
    const res = await axios.put(
      `${apiUrl}/selections/${selectionId}`,
      {
        folderId,
      },
      Util.apiPostCallSettings(cancelToken),
    );
    return res.data;
  },

  /**
   * Get one selection
   * @param {string} selectionId - id of selection
   * @param {object} cancelToken - axios token
   */
  getSelection: async (selectionId, cancelToken) => {
    const res = await axios.get(`${apiUrl}/selections/${selectionId}`, Util.apiGetCallSettings(cancelToken));
    return res.data.data;
  },

  /**
   * Save one selection
   * @param {object} postData - data of the new selection
   * @param {object} cancelToken - axios token
   */
  saveSelection: async (postData, cancelToken) => {
    const res = await axios.post(`${apiUrl}/selections/save`, postData, Util.apiPostCallSettings(cancelToken));
    return res.data;
  },
```

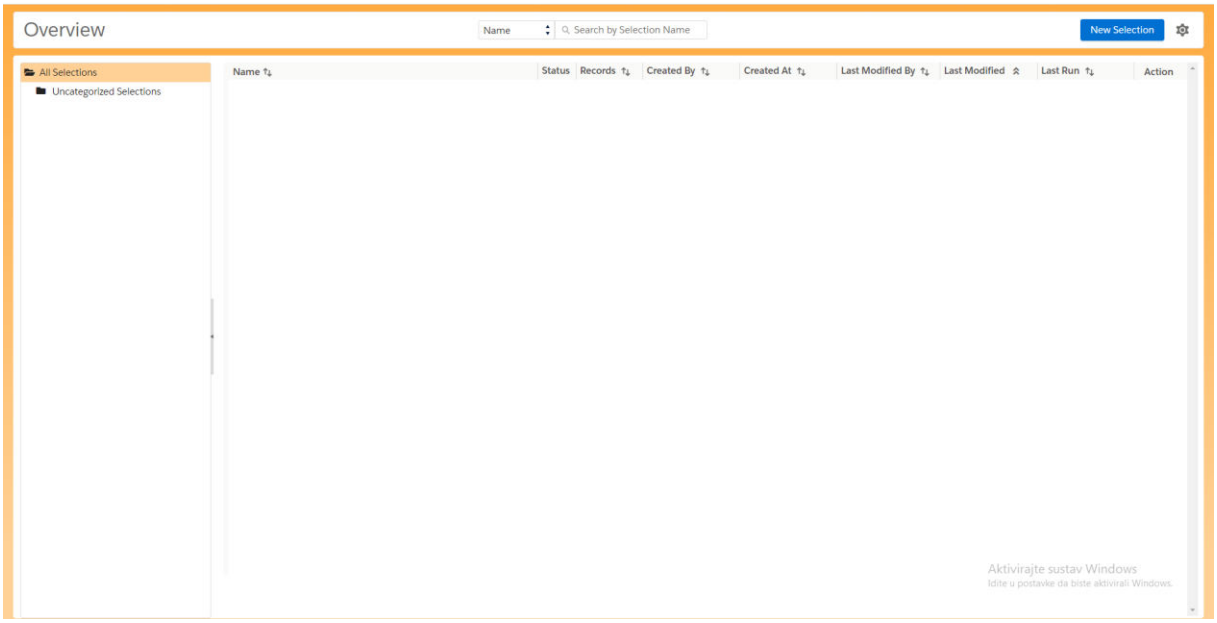
Sl. 4.63. Prikaz API funkcija za rad sa selekcijama

5. Način korištenja i ispitivanje rada ostvarene okoline

U ovom djelu će se prikazati rad same okoline, njeno korištenje i funkcionalnost. Također će se prikazati sveukupni koraci koje korisnik prolazi prilikom rada okoline te mogućnosti koje okolina nudi.

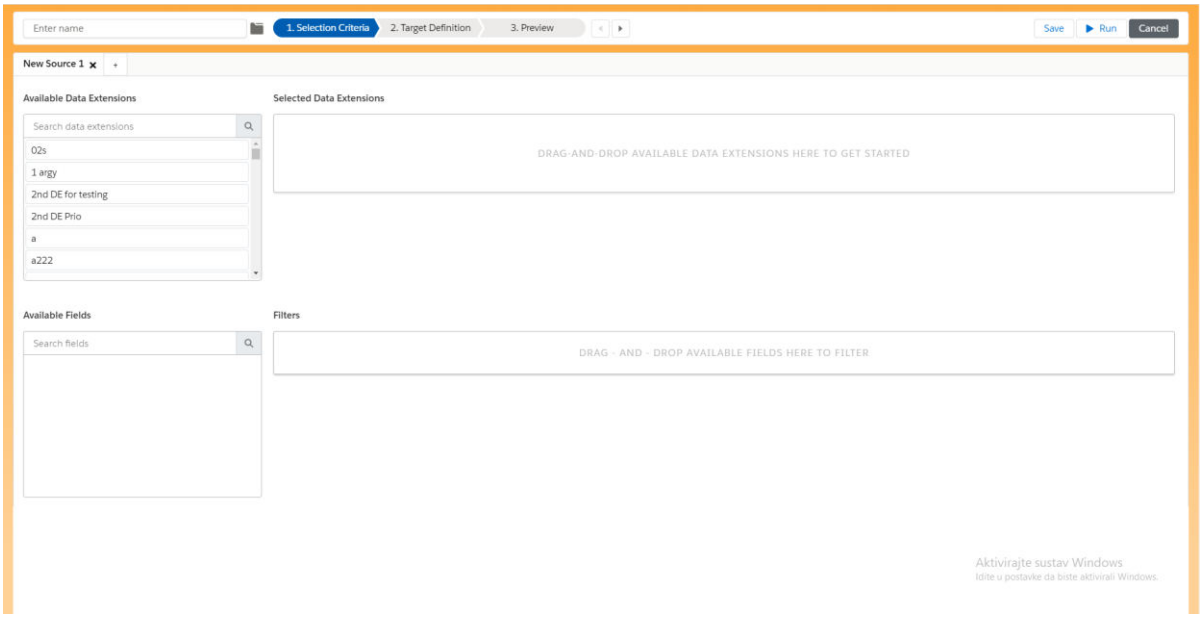
5.1. Način korištenja ostvarene okoline

Korisničko sučelje okoline pruža tri glavna koraka za stvaranje SQL upita. Svaki korak sadrži vlastite korake i funkcionalnosti koje pridonose stvaranju SQL upita. Prilikom pokretanje okoline korisniku će se prikazati početni zaslon odnosno stranica Overview na kojoj korisnik može pronaći informacije o stvorenim SQL upitima, stvarati nove upite te ih organizirati u datoteke. Početni zaslon okoline prikazan je na slici 5.1.



Sl. 5.1. Početni zaslon okoline

Prema slici 5.1 vidljivi su elementi početnog zaslona. S lijeve strane možemo vidjeti odjeljak s datotekama u koje korisnik može spremiti novonastale SQL upite. U gornjem desnom kutu slike nalazi se tipka „New Selection“. Pritiskom na tu tipku korisniku se omogućuje kreiranje SQL upita kroz tri glavna koraka. Kreirani SQL upit u sklopu okoline opisane u ovom diplomskom radu zvat ćemo Selekcija. Nakon što korisnik klikne na dugme za stvaranje nove Selekcije prikazat će se novi zaslon za stvaranje SQL upita koji se sastoji od tri glavna koraka (Slika 5.2).

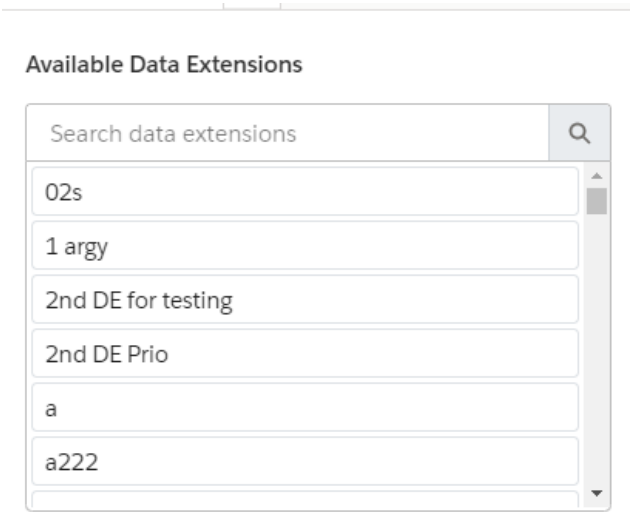


Sl. 5.2. Prikaz zaslona za stvaranje SQL upita

Prema slici 5.2 stvaranje SQL upita sastoji se od tri glavna koraka, a to su: Kriterij Selekcije (Selection Criteria), Definicija ciljnog proširenja (Target Definition) i Pregled podataka (View).

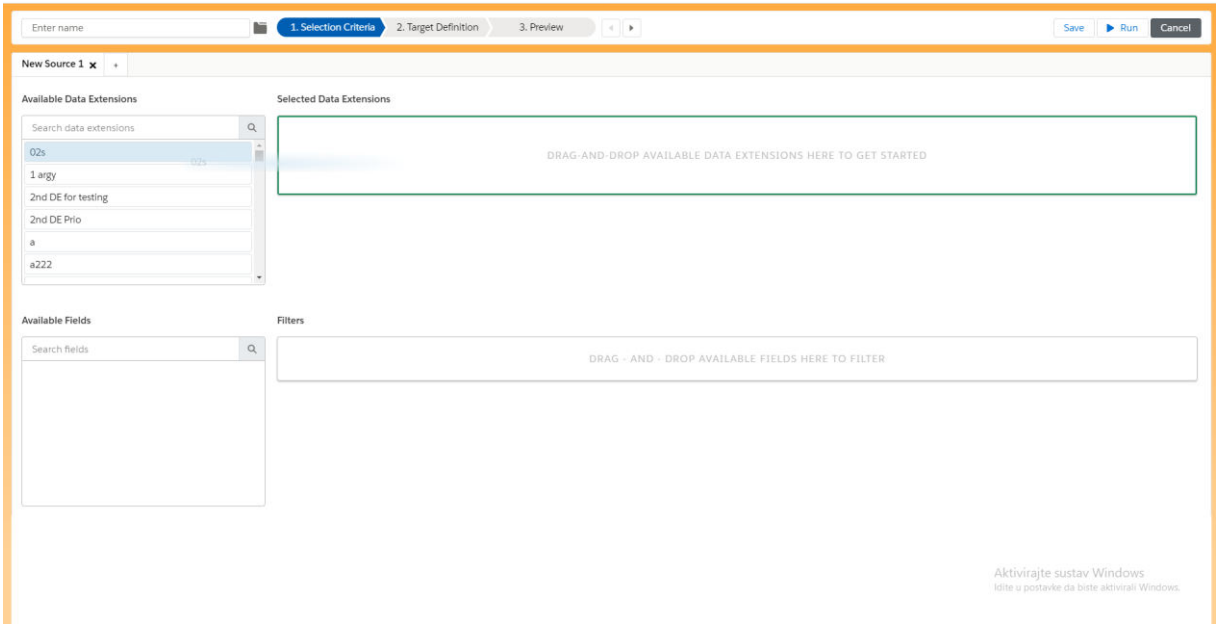
5.1.1. Kriterij Selekcije

U ovom koraku odabire se podatkovno proširenje na kojem želimo izvršiti upit te pripadajuća polja koje možemo koristiti pri filtriranju. Podatkovna proširenja prikazane su u odjeljku dostupnih podatkovnih proširenja (Slika 5.3).



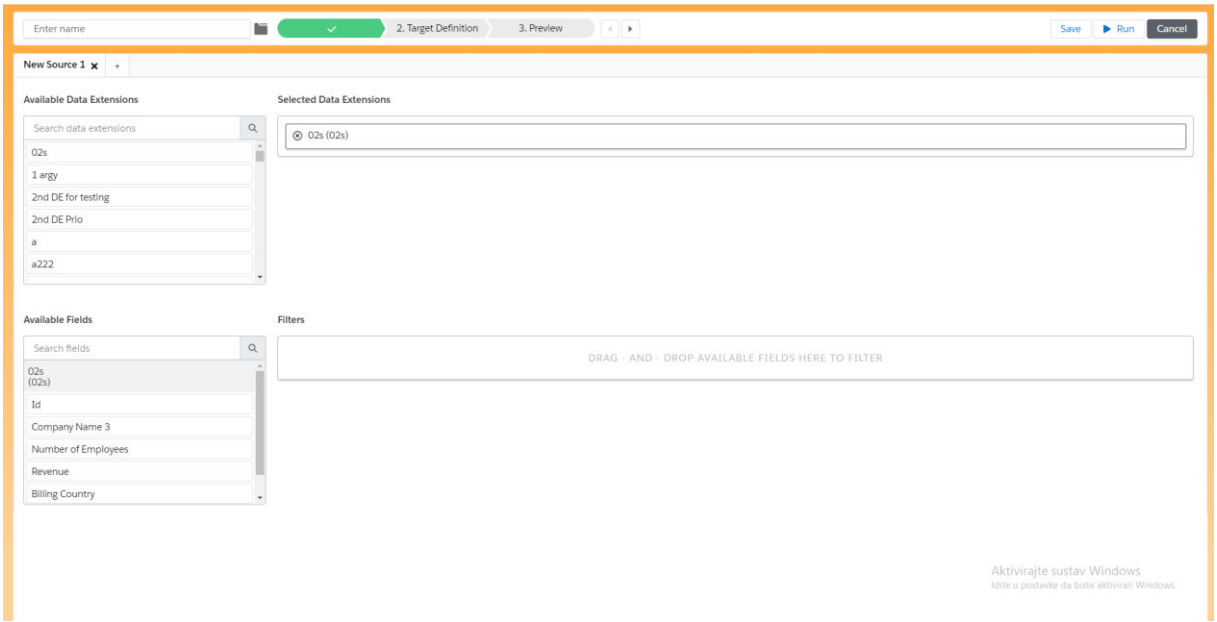
Sl. 5.3. Odjeljak dostupnih podatkovnih proširenja

Podatkovna proširenja se mogu pretraživati te pomoću metode „*drag and drop*“ premjestiti u područje odabranog podatkovnog proširenja nakon čega se prikažu i polja odabranog podatkovnog proširenja (Slike 5.4 i 5.5).



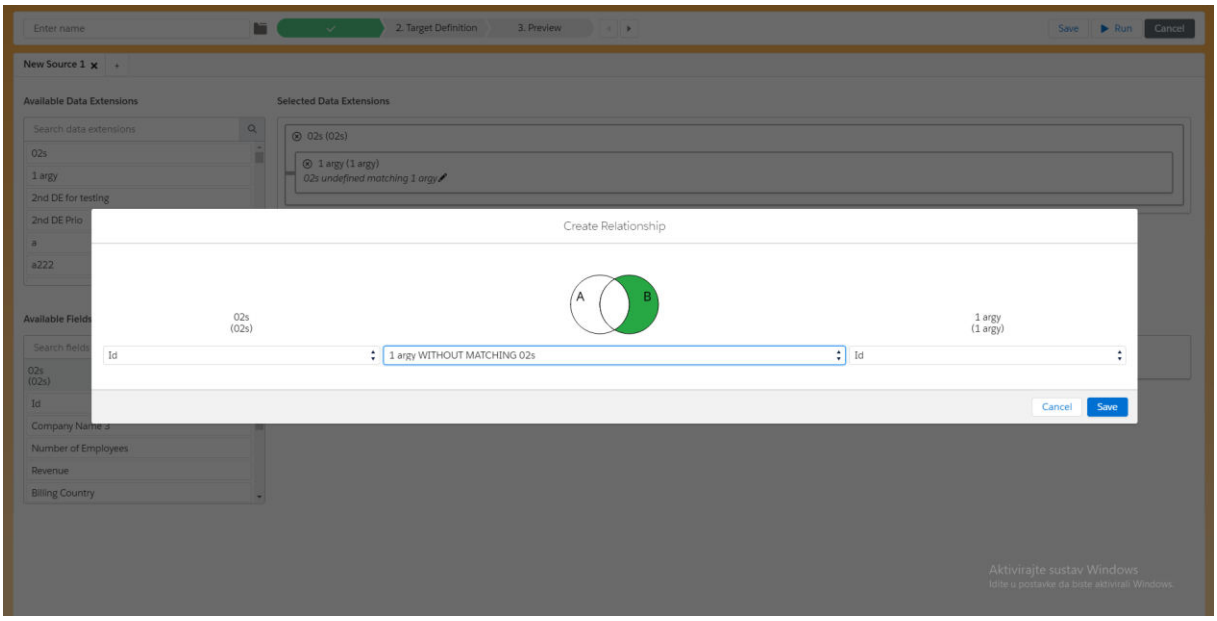
Sl. 5.4. Početak „drag and drop“ metode na odabranom podatkovnom proširenju

Prema slici 5.4 vidljivo je kako se prilikom početka metode „*drag and drop*“ obrub za predviđeno mjesto za postavljanje odabranog proširenja oboji u zelenu boju. Nakon što se ispusti podatkovno proširenje prikazuju se njegova dostupna polja u odjeljku dostupnih polja (Slika 5.5).



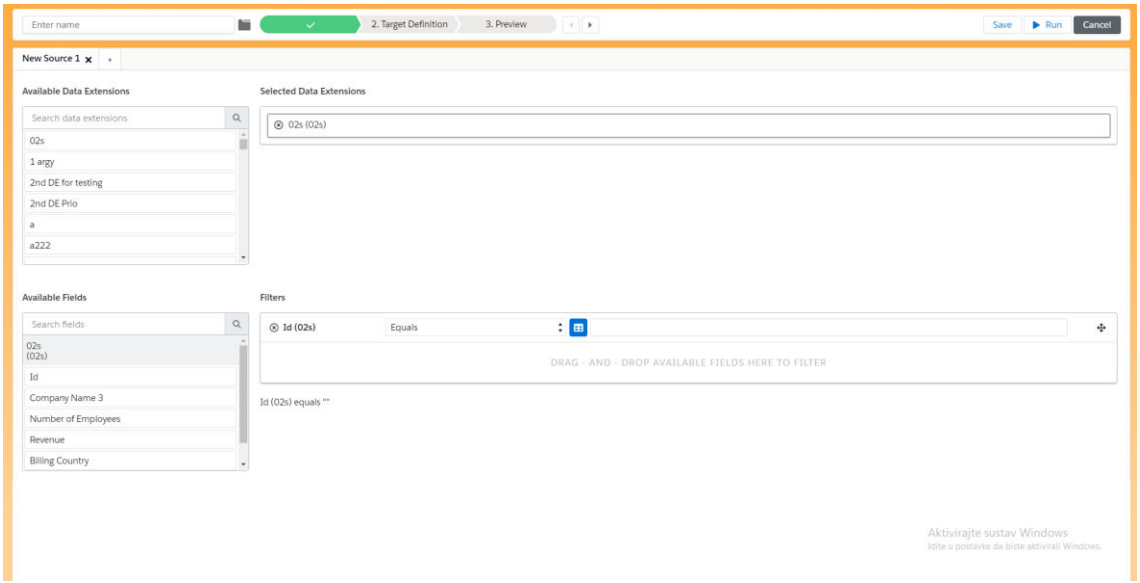
Sl. 5.5. Prikaz odabranog podatkovnog proširenja i njegovih polja

Želimo li povezati više podatkovnih proširenja jednostavno metodom „*drag and drop*“ odaberemo drugo podatkovno proširenje i ispustimo ga na postojeće nakon čega će se pojaviti modal u kojemu je potrebno definirati polja koja ih povezuju te odnos između podatkovnih proširenja (Slika 5.7).



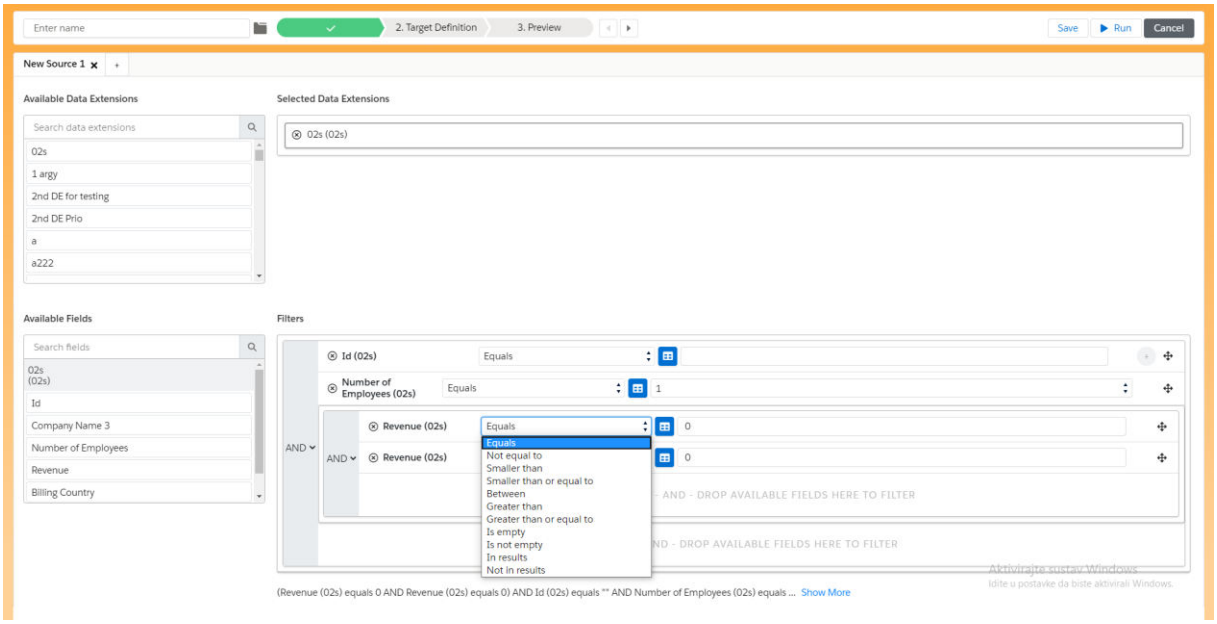
Sl. 5.7. Prikaz stvaranja odnosa između podatkovnih proširenja

Nakon odabira željenih podatkovnih proširenja te definiranja odnosa između njih stvaraju se filtri. Filtre možemo stvoriti pomoću metode „*drag and drop*“ nad poljima podatkovnog proširenja. Polje koje želimo koristiti za filtriranje odabranog podatkovnog proširenja metodom „*drag and drop*“ ispustimo u odjeljak za filtre. Prikaz stvorenog filtra nalazi se na slici 5.8.



Sl. 5.8. Prikaz filtra

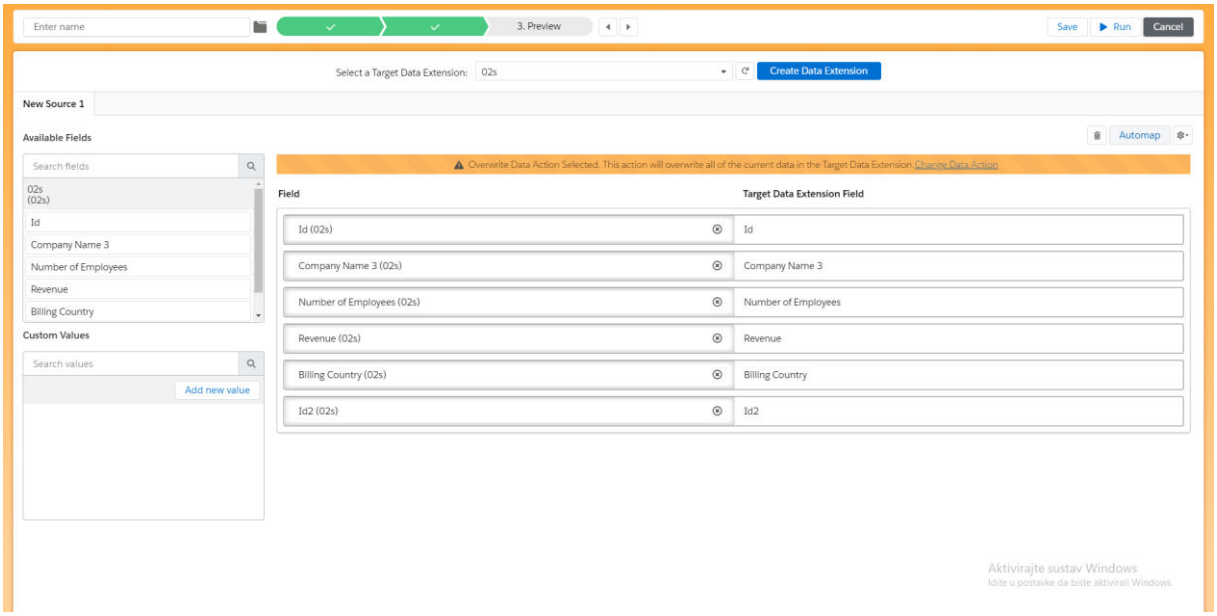
Prema slici 5.8 vidi se kako je kreiran filter pomoću polja Id te je početna vrijednost kriterija filtra jednaka „*Equal*“ što znači da će se vrijednost polja Id uspoređivati s vrijednosti koju unesemo u polja za unos. Također, tekst cijelog filtra prikazan je ispod samog odjeljka za filtre. Moguće je kombinirati više filtra, spajati ih te mijenjati kriterije (Slika 5.9).



Sl. 5.9. Prikaz dostupnih kriterija za filtriranje

5.1.2. Definicija ciljnog proširenja

U ovom koraku definira se podatkovno proširenje koje će podaci nastali SQL upitom biti spremljeni. Podatkovno proširenje odabiremo iz padajućeg izbornika u kojem se nalaze sva dostupna proširenja. Nakon što smo odabrali željeno proširenje potrebno je pridružiti polja iz izvorišnog podatkovnog proširenja određenim poljima odabranog proširenja. Vrijednosti pridruženih polja, nakon završetka SQL upita, spremaju se u polja ciljnog podatkovnog proširenja (Slika 5.10).



Sl. 5.10. Prikaz definicije ciljnog podatkovnog proširenja

5.1.3. Pregled podataka

Nakon što su se odabrala podatkovna proširenja iz kojih će se izvlačiti podaci i u koje će se ti podaci spremati, definirali filteri te pridružili polja za stvaranje SQL upita, pritiskom na tipku „Run Preview“ prikazat će se rezultati nastale izvršavanjem SQL upita (Slika 5.11).

Sl. 5.11. Prikaz sučelja za prikaz rezultata SQL upita

5.2. Primjeri stvaranja složenih SQL upita

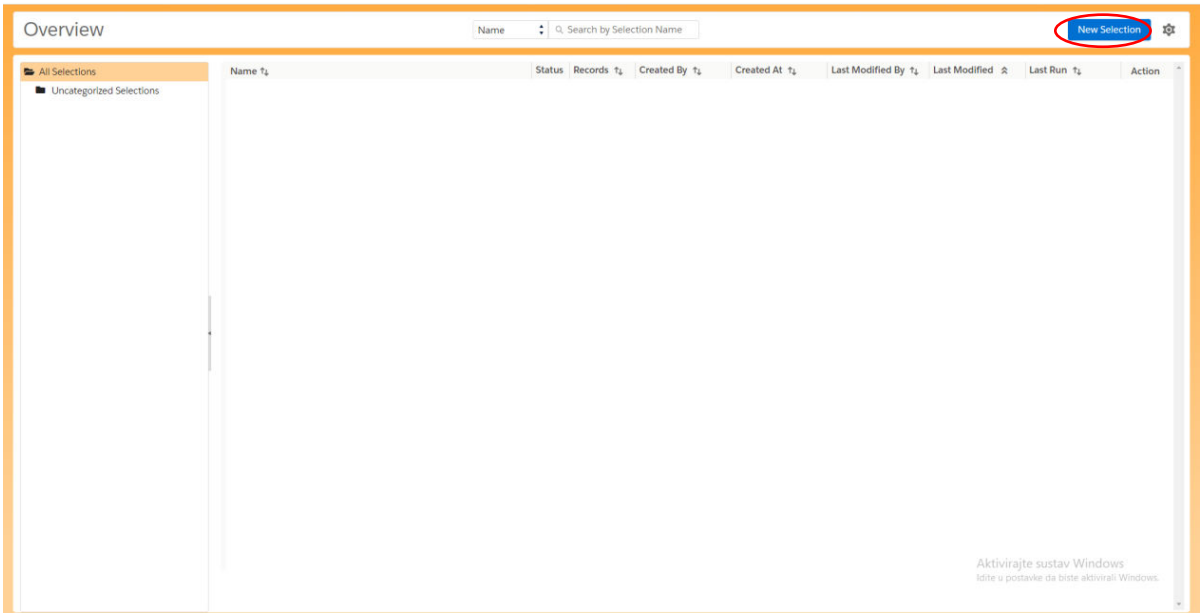
U ovom poglavlju opisat će se stvaranje složenih SQL upita te na primjerima prikazati funkcionalnosti okoline. Najprije su stvorena ciljna podatkovna proširenja unutar Salesforce Marketing Cloudu pomoću kojih će se pokazati različiti primjeri stvaranje SQL upita od jednostavnih do složenih. Stvorena podatkovna proširenja s pripadajućim poljima prikazana su u tablici 5.1:

Tablica 5.1. Prikaz podatkovnih proširenja

Ime	**Opis**	**Polja**
DESELECT_DEMO_PRODUCTS	Sadrži popis proizvoda koje bi tvrtka mogla prodati	Id, Name, List Price, Product Family
DESELECT_DEMO_ORDERS	Sadrži narudžbe kupca	Id, Date, AccountId,ContactId
DESELECT_DEMO_ORDERLINES	Sadrži linije narudžbi koje pripadaju narudžbama iz podatkovnog proširenja DESELECT_DEMO_ORDERS	Id, OrderId, ProductId, Sales Price, Amount
DESELECT_DEMO_ACCOUNTS	Sadrži podatke o tvrtki kupca	Id, Company Name, Number of Employees, Revenue, Billing Contry
DESELECT_DEMO_CONTACTS	Sadrži kontakt podatke o osobama koji rade u tvrtkama u podatkovnom proširenju DESELECT_DEMO_ACCOUNTS	Id, AccountId, First Name, Last Name, Email
DESELECT_DEMO_EMAILS TO EXCLUDE	Sadrži popis e-adresa koje bi se mogle koristiti za popise izuzeća, npr. kada se želi poslati e-pošta kupcima, ali izuzeti neke e-adrese.	Email

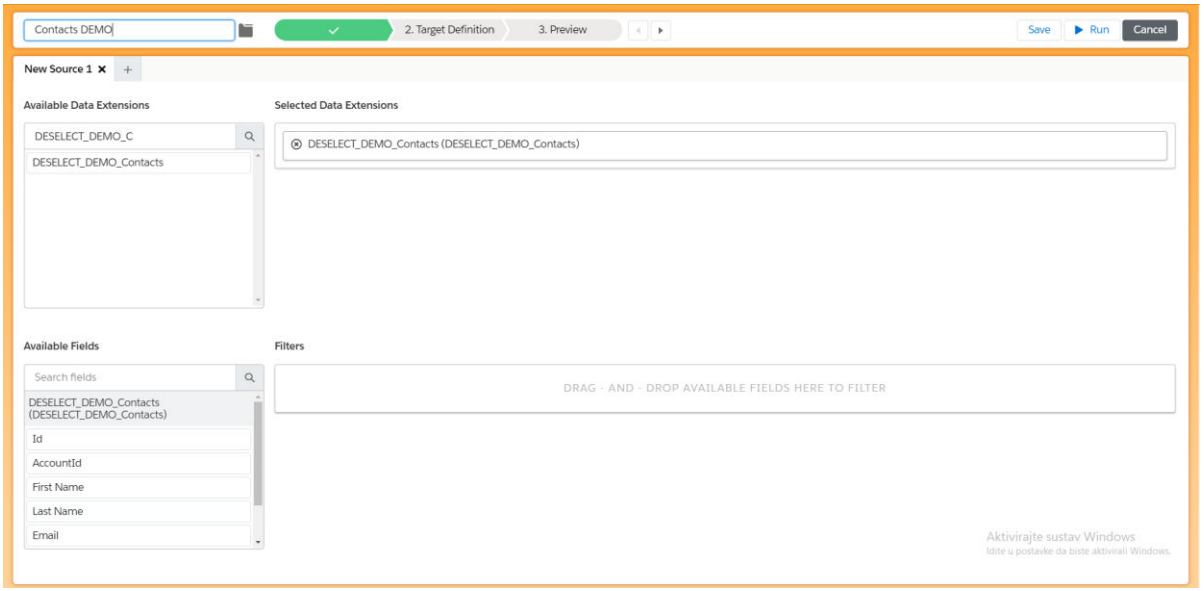
Primjer 1. Odaberi kontakte s pripadajućim imenom tvrtki

Na zaslonu s pregledom odabira potrebno je kliknuti na tipku „New Selection“ kako bi se započelo stvaranje nove selekcije (Slika 5.12).



Sl. 5.12. Prikaz odabira tipke „New Selection“

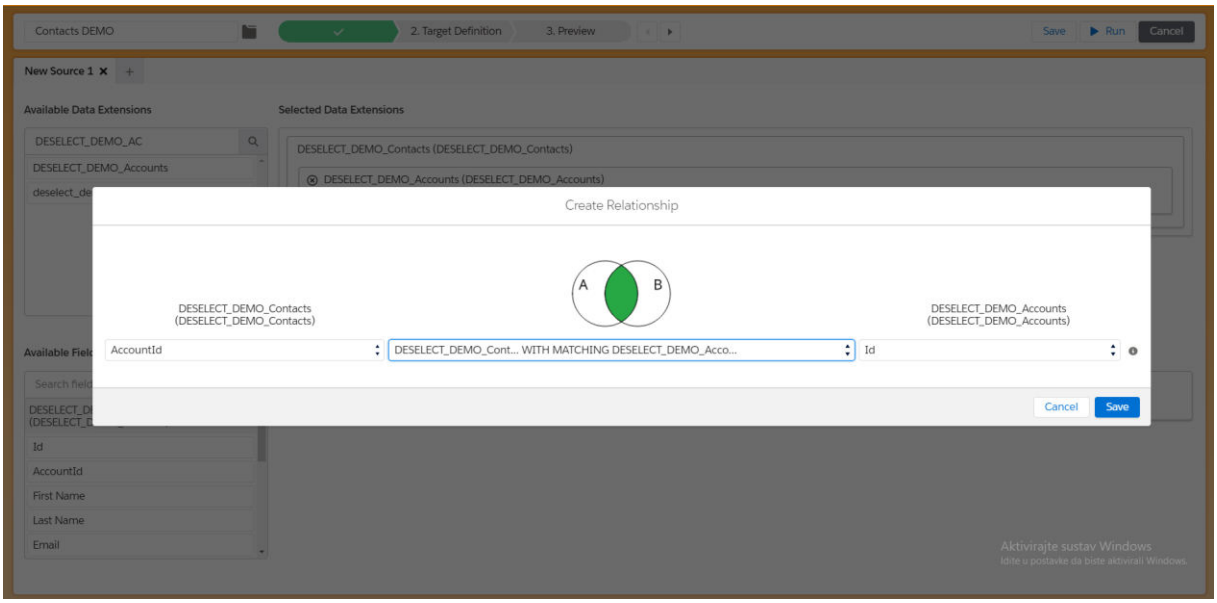
Pomoću metode „*drag and drop*“ podatkovno proširenje DESELECT_DEMO_Contacts povlači se u odjeljak Odabrana proširenja podataka (Slika 5.13).



Sl. 5.13. Prikaz odabira DESELECT_DEMO_CONTACTS podatkovnog proširenja

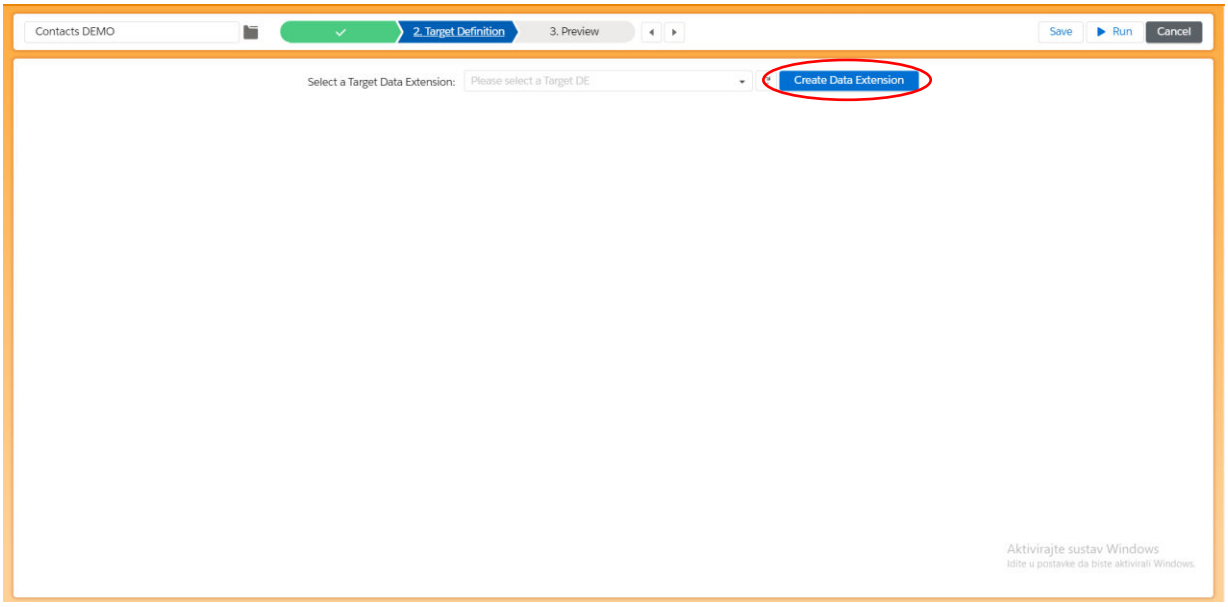
Nakon toga potrebno je povući metodom „*drag and drop*“ podatkovno proširenje DESELECT_DEMO_Accounts na vrh podatkovnog proširenja DESELECT_DEMO_Contacts u odjeljku Odabrana proširenja podataka. Sljedeći korak je definiranje odnosa između odabranih podatkovnih proširenja. Odnos između odabranih podatkovnih proširenja definira se tako što se odabiru polja AccountId iz podatkovnog

proširenja DESELECT_DEMO_Contacts te polje Id iz podatkovnog proširenja DESELECT_DEMO_Accounts te se povezuju s odnosom DESELECT_DEMO_Contacts WITH MATCHING DESELECT_DEMO_Accounts (Slika 5.14)

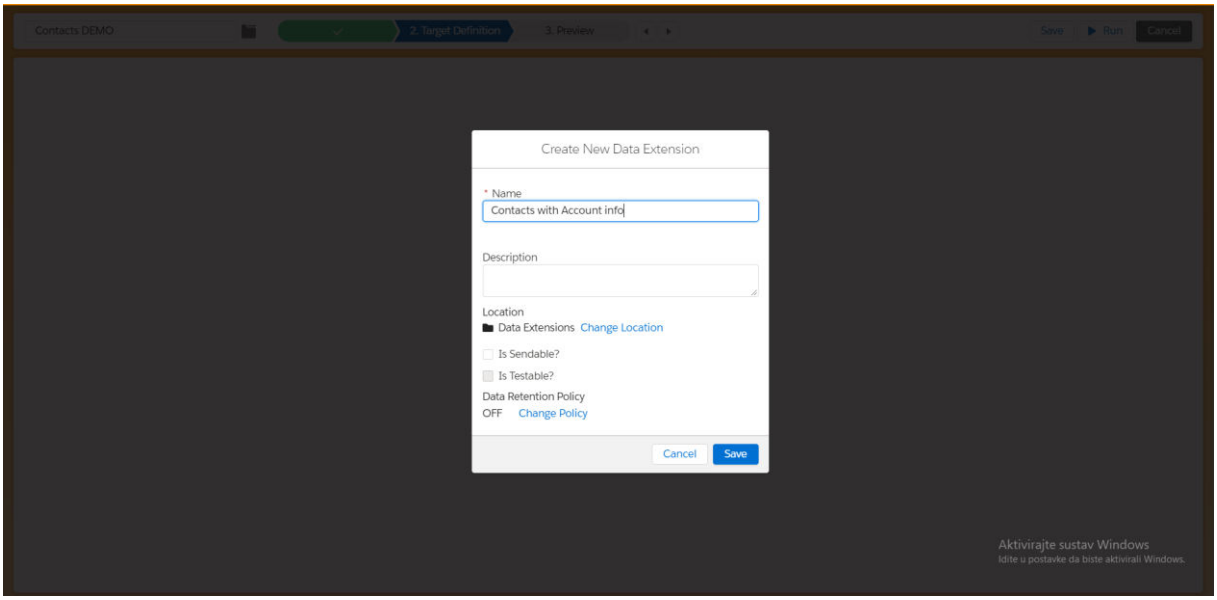


Sl. 5.14. Prikaz stvaranja odnosa između DESELECT_DEMO_Contacts i DESELECT_DEMO_Accounts podatkovnih proširenja

Sljedeći korak je definiranje ciljnog podatkovnog proširenja. Pritiskom na tipku Create Data Extension, otvara se novi model gdje je potrebno unijeti naziv podatkovnog proširenja, npr. Contacts with Account info. Pritiskom na tipku Save privremeno se spremaju unijete informacije o podatkovnom proširenju (Slika 5.15 i Slika 5.16).

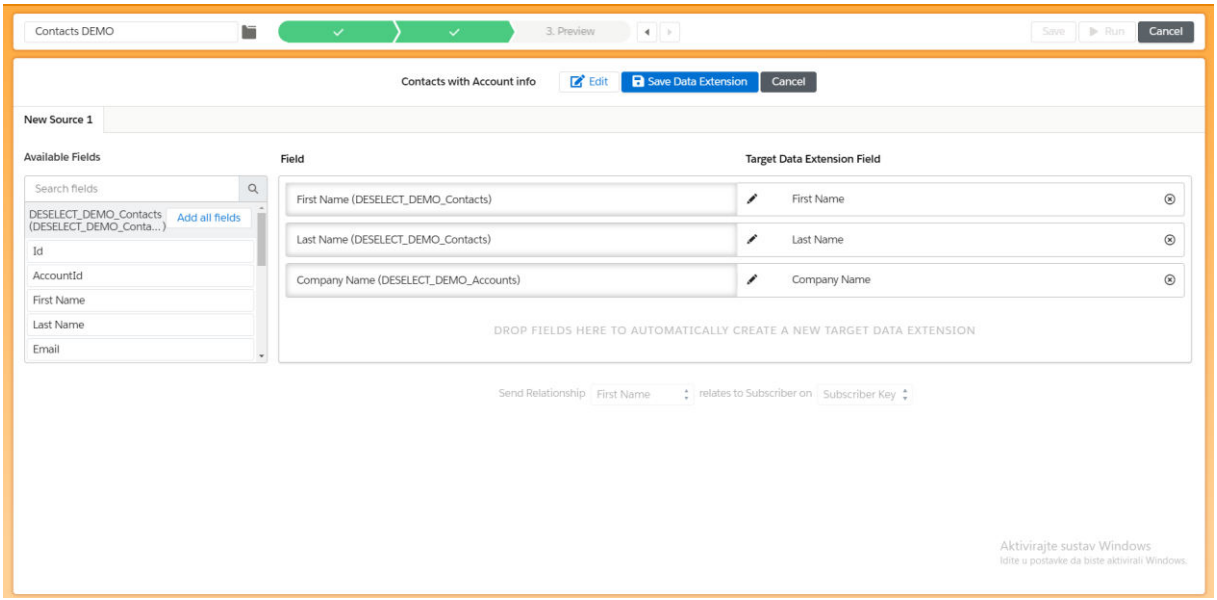


Sl. 5.15. Prikaz pritiska na tipku Create Data Extension



Sl. 5.16. Prikaz unosa podataka o novom podatkovnom proširenju

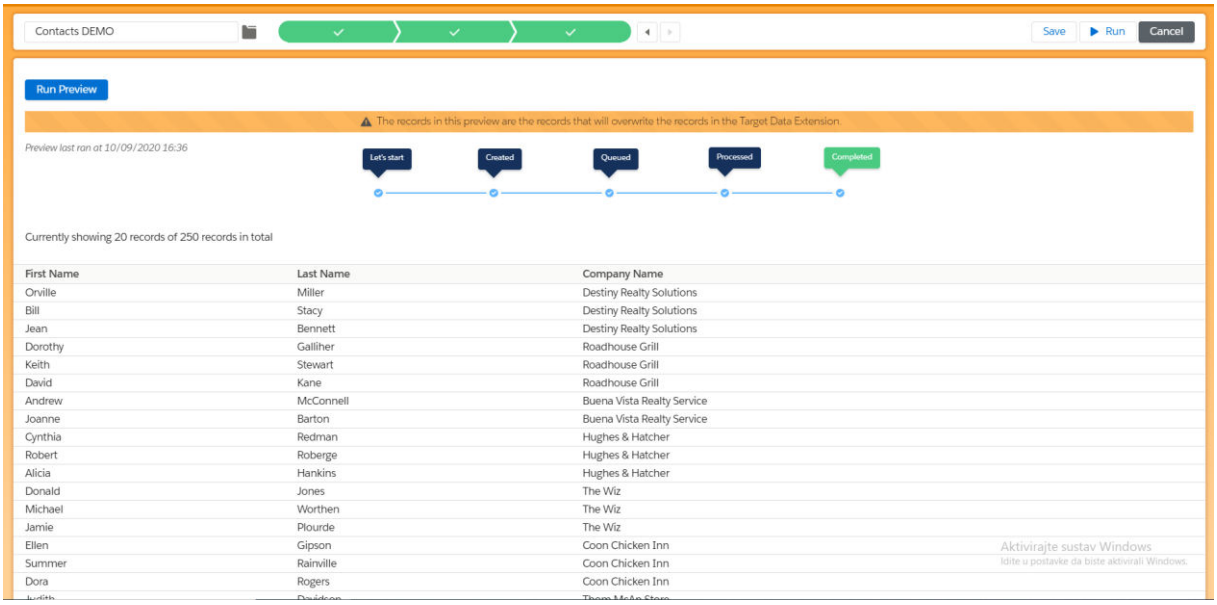
Prilikom stvaranja novog podatkovnog proširenja potrebno je odabrati sljedeća polja dvostrukim klikom na njih ili povlačenjem u odjeljak s desne strane (Slika 5.17): First Name, Last Name iz podatkovnog proširenja DESELECT_DEMO_Contacts i Company Name iz podatkovnog proširenja DESELECT_DEMO_Accounts. Pritiskom na tipku Save Data Extension stvara se novo podatkovno proširenje unutar Salesforce Marketing Clouda.



Sl. 5.17. Prikaz stvaranja novog podatkovnog proširenja

Kada se na zaslonu Pregled klikne Pokreni pregled, prikazat će se tablica rezultata s poljima First Name, Last Name i Company Name (Slika 5.18). SQL upit nastao korištenjem okoline kroz opisane korake je sljedeći: "SELECT "DESELECT_DEMO_Contacts"."First Name" AS

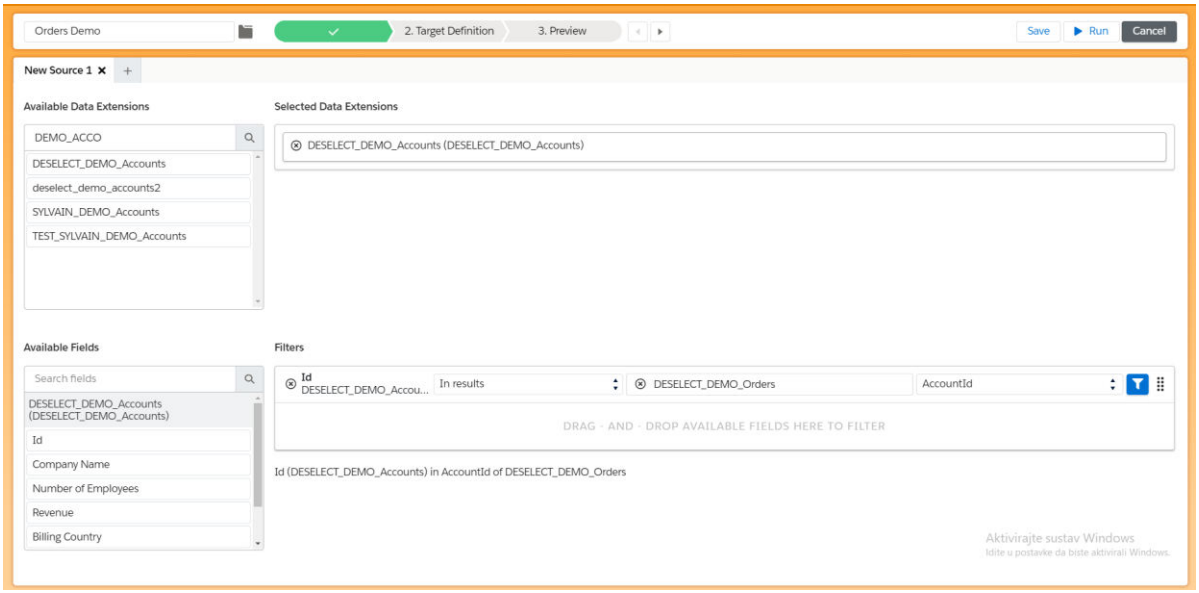
"First Name", "DESELECT_DEMO_Contacts"."Last Name" AS "Last Name", "DESELECT_DEMO_Accounts"."Company Name" AS "Company Name" FROM "DESELECT_DEMO_Contacts" "DESELECT_DEMO_Contacts" INNER JOIN "DESELECT_DEMO_Accounts" "DESELECT_DEMO_Accounts" ON "DESELECT_DEMO_Contacts"."AccountId" = "DESELECT_DEMO_Accounts"."Id""



Sl. 5.18. Prikaz tablice s novonastalim rezultatima

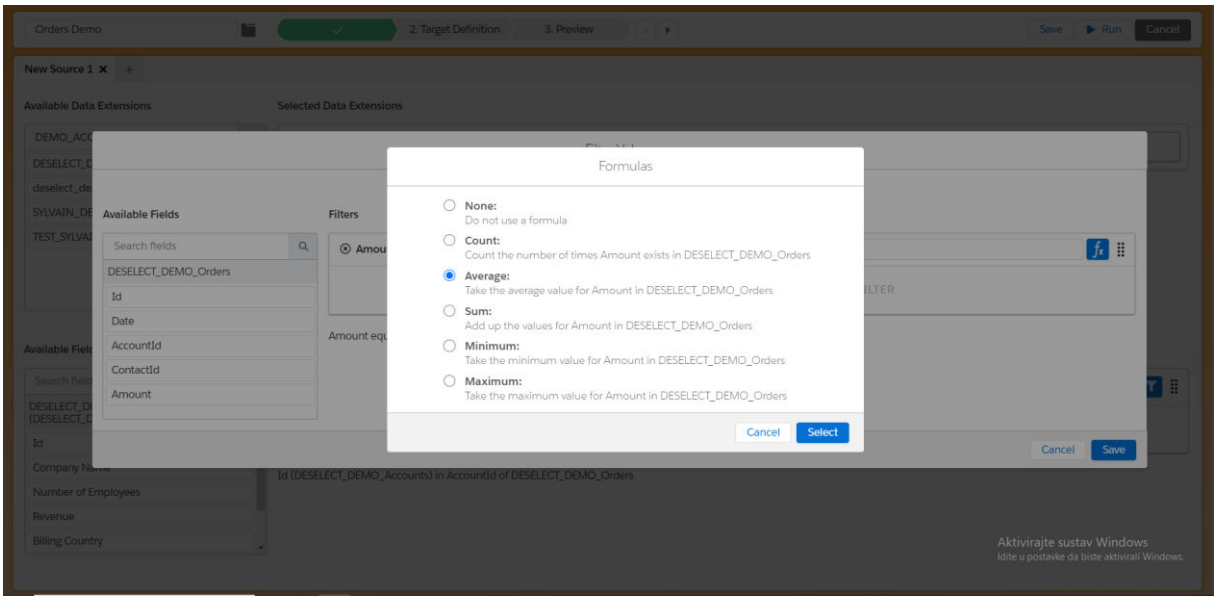
Primjer 2. Odaberi račune čija je prosječna veličina narudžbe veća od 50 eura

Na zaslonu s pregledom odabira potrebno je kliknuti na tipku „New Selection“ kako bi se započelo stvaranje nove selekcije (Slika 5.12). Pomoću metode „drag and drop“ potrebno je podatkovno proširenje DESELECT_DEMO_Accounts povući u odjeljak Odabrana proširenja podataka (Slika 5.13). Povucite Id polje u odjeljak za filtre. Odaberite kriterij „In result“. Iz padajućeg izbornika dostupnih proširenja odaberite proširenje DESELECT_DEMO_Orders. Odaberite polje AccountId (Slika 5.19). Sada smo naznačili da želimo dobiti račune čiji se Id pojavljuje u podatkovnom proširenju narudžbe. Međutim, želimo filtrirati samo račune s prosječnom veličinom narudžbe iznad 50 eura.



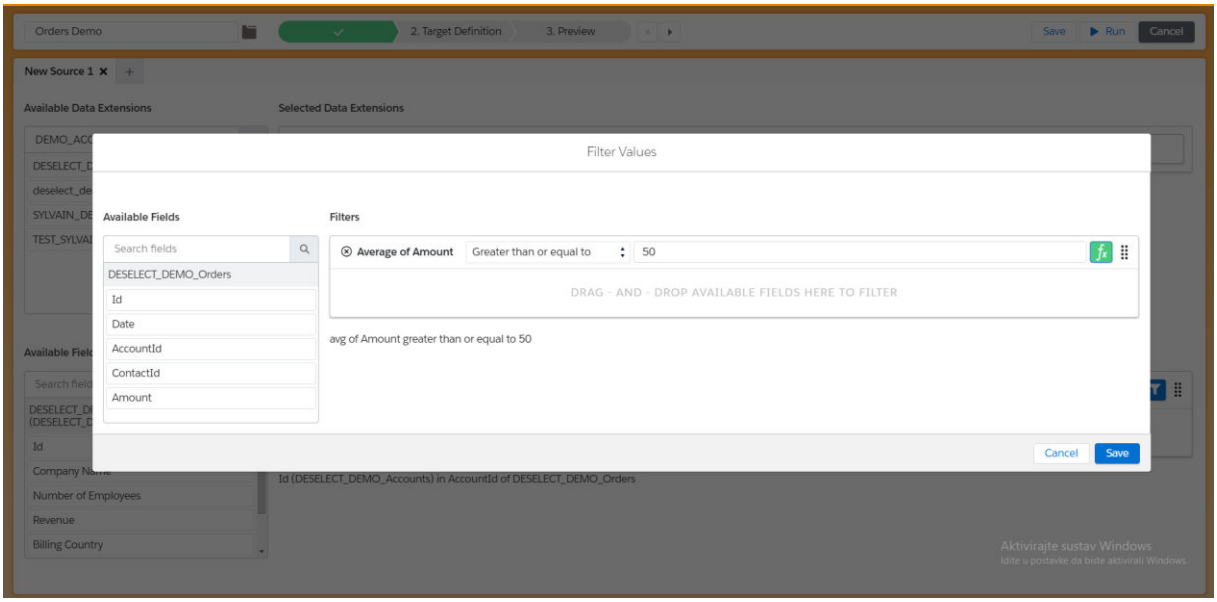
Sl. 5.19. Prikaz stvaranja filtra

Kliknite ikonu toka na novostvorenom filtru. Otvorit će se novi modal u kojemu je potrebno povuci polje Amount u odjeljak za filtre. Kliknite ikonu formule i odaberite Average (Slika 5.20).



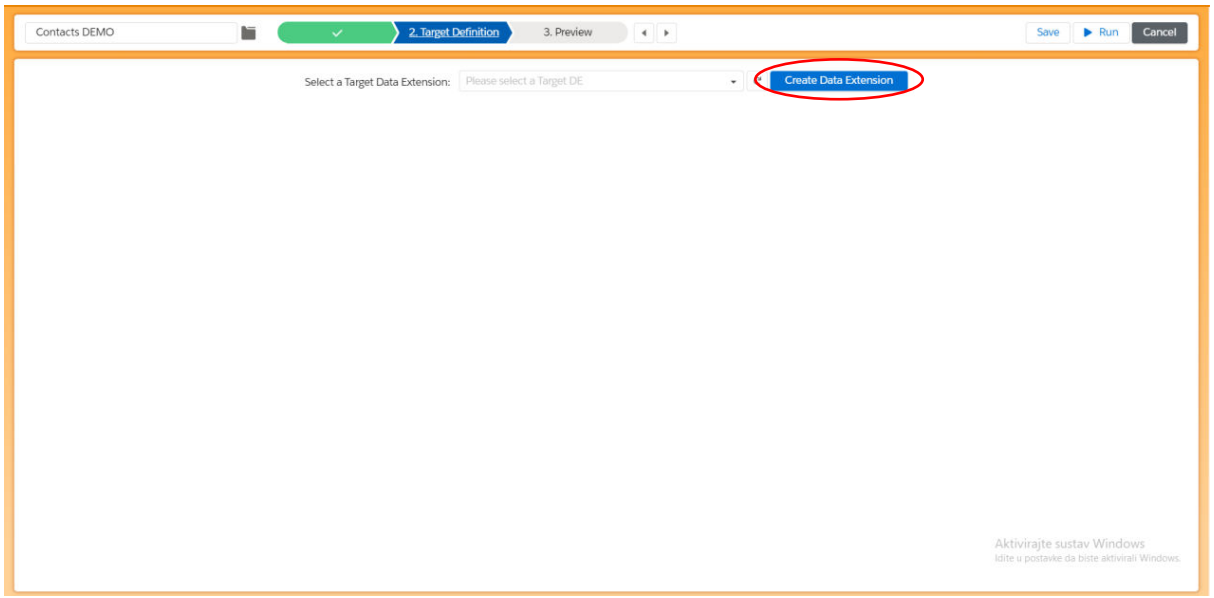
Sl. 5.20. Prikaz odabira formule Average nad poljem Amount

Nakon toga ikona formule postaje zelena, a naziv polja mijenja se u Average Amount. Odaberite kriterij: Greater than or equal to. Za vrijednost odaberite: 50 (Slika 5.21).

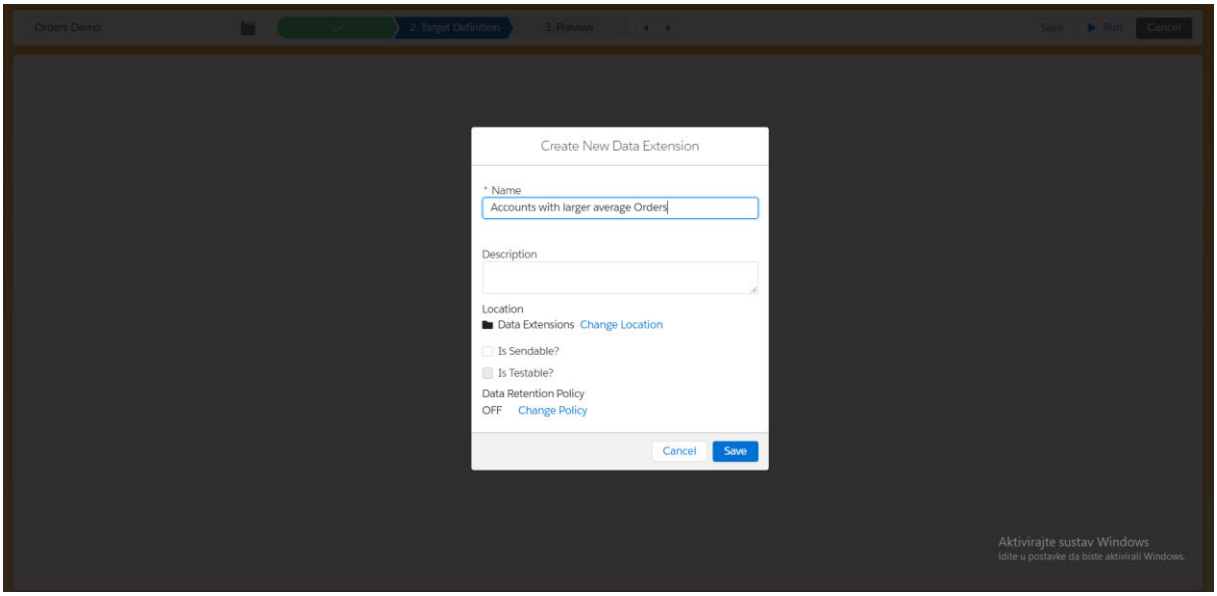


Sl. 5.21. Prikaz definiranja filtra za filtriranje računa s prosječnom veličinom narudžbe iznad 50 eura.

Pritisnite gumb Create Data Extension, unesite naziv, npr. Accounts with larger average Orders i pritisnite Save (Slika 5.22 i Slika 5.23).

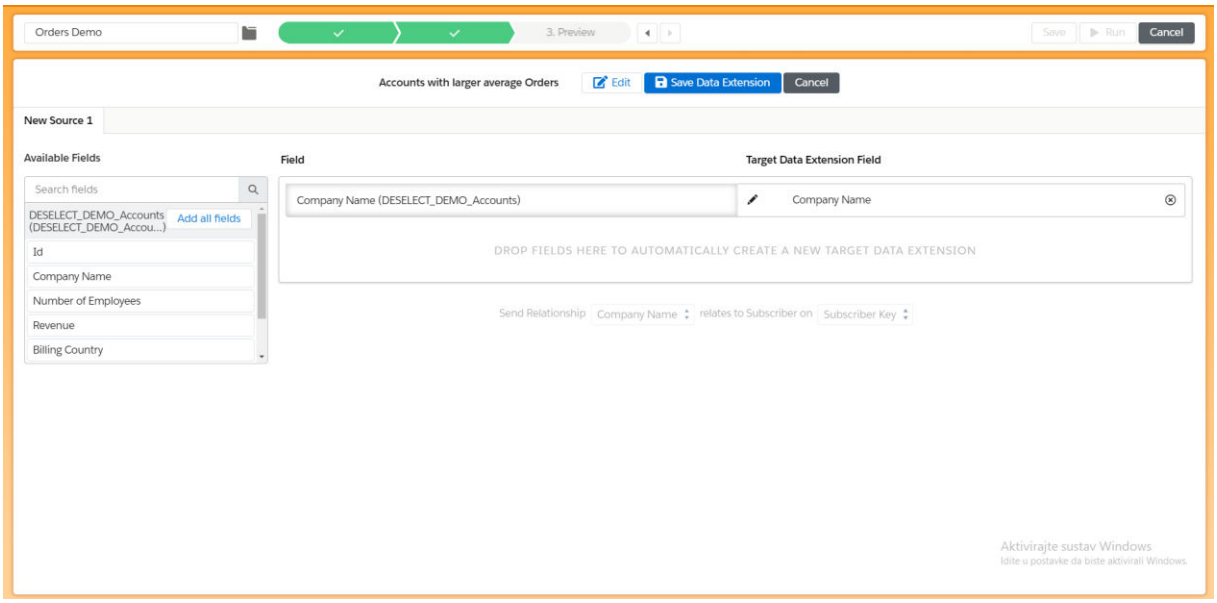


Sl. 5.22. Prikaz pritiska na tipku Create Data Extension



Sl. 5.23. Prikaz unosa podataka o novom podatkovnom proširenju

Odaberite sljedeća polja dvostrukim klikom na njih ili povlačenjem u odjeljak s desne strane (Slika 5.24): Company Name iz DESELECT_DEMO_Accounts. Kliknite Save Data Extension.



Sl. 5.24. Prikaz stvaranja novog podatkovnog proširenja

Kada na zaslonu Pregled kliknete Pokreni pregled, prikazat će se tablica rezultata s poljem Company Name (Slika 5.25). SQL upit nastao korištenjem okoline kroz opisane korake je sljedeći: "SELECT "DESELECT_DEMO_Accounts"."Company Name" AS "Company Name" FROM "DESELECT_DEMO_Accounts" "DESELECT_DEMO_Accounts" WHERE "DESELECT_DEMO_Accounts"."Id" IN (SELECT

"DESELECT_DEMO_Orders"."AccountId" AS "AccountId" FROM "DESELECT_DEMO_Orders" "DESELECT_DEMO_Orders" GROUP BY "DESELECT_DEMO_Orders"."AccountId" HAVING AVG("DESELECT_DEMO_Orders"."Amount") >= 50)".

Sl. 5.25. Prikaz tablice s novonastalim rezultatima

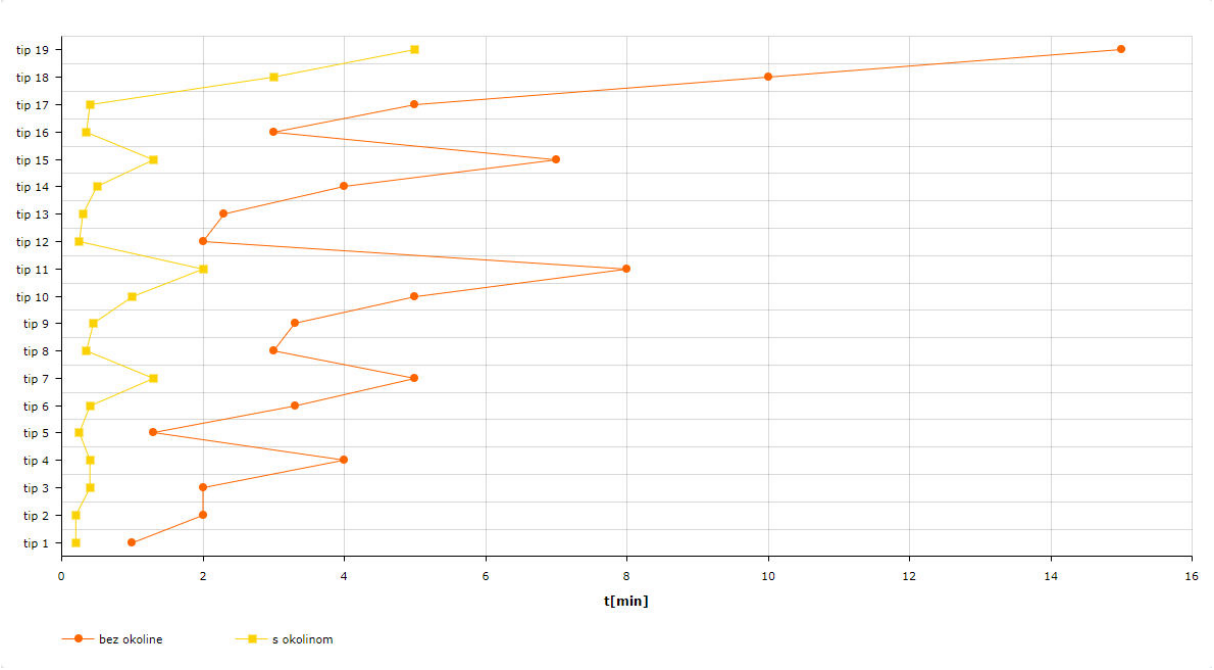
5.3. Analiza uspješnosti stvaranja SQL upita

Primjerima iz prethodne cjeline pokazano je kako se u kratkom vremenskom razdoblju kroz korisničko sučelje okoline stvaraju složeni SQL upiti. Kada bi se takvi upiti pokušali napisati ručno odnosno bez korištenja okoline razvijene u ovom diplomskom radu bilo bi potrebno nekoliko sati. Usporedba stvaranja SQL upita bez i uz pomoć okoline razvijene u ovom diplomskom radu prikazana je uz korištenje tablice 5.2 i slike 5.26. U tablici 5.2 dani su tipovi SQL upita. Tablica 5.2 prikazuje različite SQL upite čija su stvaranja vremenski prikazana grafom na slici 5.26. Na slici 5.26 vidljiva je razlika u vremenima stvaranja SQL upita uz korištenje i bez korištenje okoline.

Tablica 5.2. Prikaz tipova SQL upita

Tip upita	**SQL upit**
Tip 1	SELECT FROM (5 polja, 1 podatkovno proširenje)
Tip 2	SELECT FROM (10 polja, 1 podatkovno proširenje)

Tip 3	SELECT FROM + UNION (5 polja, 1 podatkovno proširenje za svaki UNION)
Tip 4	SELECT FROM + UNION (10 polja, 1 podatkovno proširenje za svaki UNION)
Tip 5	SELECT FROM + WHERE (1 filter)
Tip 6	SELECT FROM + WHERE (5 filtera)
Tip 7	SELECT FROM + WHERE (10 filtera)
Tip 8	SELECT FROM + WHERE + podupit
Tip 9	SELECT FROM + WHERE + podupit (1 filter)
Tip 10	SELECT FROM + WHERE + podupit (5 filtera)
Tip 11	SELECT FROM + WHERE + podupit (10 filtera)
Tip 12	SELECT FROM s 2 povezana podatkovna proširenja
Tip 13	SELECT FROM + WHERE (1 filter) s 2 povezana podatkovna proširenja
Tip 14	SELECT FROM + WHERE (5 filtera) s 2 povezana podatkovna proširenja
Tip 15	SELECT FROM + WHERE (10 filtera) s 2 povezana podatkovna proširenja
Tip 16	SELECT FROM + WHERE + podupit s 2 povezana podatkovna proširenja
Tip 17	SELECT FROM s 4 povezana podatkovna proširenja
Tip 18	SELECT FROM + WHERE (s više od 5 filtera) s 4 povezana podatkovna proširenja
Tip 19	SELECT FROM + WHERE + podupit (s više od 5 filtera) s 4 povezana podatkovna proširenja

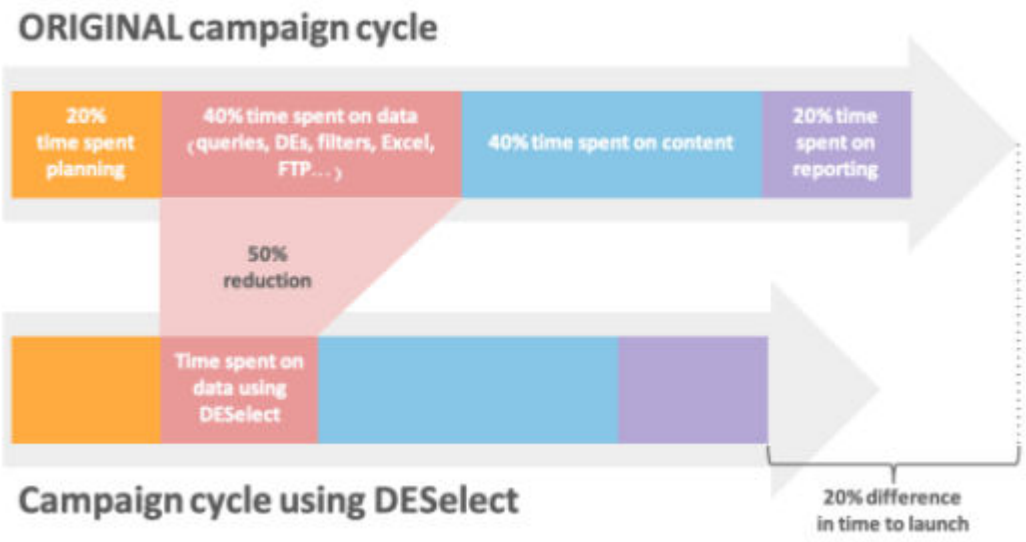


Sl. 5.26. Prikaz vremenskog trajanja stvaranja SQL upita s i bez korištenje okoline

Prema slici 5.26 vidljivo je kako je vrijeme stvaranja SQL upita uz korištenje okoline razvijene u ovom diplomskom radu znatno manje u odnosu na vrijeme stvaranja SQL upita bez korištenje okoline. Također je vidljivo kako se vrijeme stvaranja SQL upita povećava u ovisnosti o veličini polja podatkovnog proširenja, broju podatkovnih proširenja i složenosti SQL upita. Naravno, treba uzeti u obzir i vrijeme potrebno za odabir polja podatkovnog proširenja, pripremu i odabir izvorišnih i ciljnih podatkovnih proširenja te vrsti filtra korištenog prilikom stvaranja SQL upita. Uzevši u obzir navedeno, razlika u vremenu stvaranja SQL upita s i bez okoline postaje još veća.

Važno je napomenuti da su SQL upiti nastali korištenjem okoline bez pisanja SQL koda što znači da korisnici ne moraju znati SQL jezik. S obzirom na činjenicu da korisnici ne moraju znati SQL jezik tvrtke koje bi koristili ovu okolinu ne bi morale ulagati novac u tehničko osoblje za razvijanje SQL upita. Marketinški stručnjaci troše 40% svog vremena na pripremu podataka za promidžbu: stvaranje podatkovnih proširenja, filtriranje, uvoz / izvoz podataka, čišćenje podataka u Excelu, izuzeće / suzbijanje kontakata iz kampanja, upotreba FTP protokola, upotreba vanjskog alata za složenu segmentaciju i određivanje prioriteta pravila. Osim toga marketinški stručnjaci često ovise o vanjskim resursima za pisanje SQL upita kako bi se podaci pripremili za promidžbu [37]. Osim pripreme podataka za promidžbu, marketinški stručnjaci moraju voditi brigu o planiranju marketinške strategije te analizi tržišta, eksperimentiranju i ostalih bitnih stvari za ostvarivanje uspješne promidžbe [38].

Korištenjem okoline razvijene u ovom diplomskom radu dokazano je kako je vrijeme potrebno za pripremu podataka koje marketinški stručnjaci potroše smanjeno s 40% na 20% ukupnog vremena (slika 5.27).



Sl. 5.27. Prikaz odnosa trajanja pripreme podataka uz i bez korištenja okoline

S obzirom da se vrijeme na pripremu podataka prepolovilo, marketinški stručnjaci imaju više vremena za planiranje marketinške strategije, analizu tržišta, istraživanje, odnosno vjerojatnost za uspješnost promidžbe se na taj način povećala.

5.4. Povratne informacije korisnika

U ovom poglavlju predstavit će se povratne informacije korisnika, njihove priče o poteškoćama vezanih za pripremu podataka za promidžbu te način na koji im je okolina razvijena o ovom diplomskom radu pomogla u rješavanju tih poteškoća.

5.4.1. Cambridge University Press (CUP)

CUP počeo je koristiti Salesforce Marketing Cloud u lipnju 2018., ali nije uspio u potpunosti iskoristiti snagu Salesforce Marketing Cloud okoline [39]. Kao što Cherry Otto, voditeljica marketinga podataka u CUP-u objašnjava: „Naša strategija segmentacije uključuje postupak redovitog prijenosa podataka iz četiri različita sustava u Salesforce Marketing Cloud. Ti sustavi uključuju Salesforce Sales Cloud i CRM treće strane koji sadrže podatke o školskim kontaktima iz Velike Britanije. Međutim, morali bismo ručno kombinirati ove izvore podataka da bismo došli do skupa podataka koji bismo mogli koristiti za kampanje.“

Ručno kombiniranje tih izvora podataka (i njihov prijenos u ono što je u Salesforce Marketing Cloud-u poznato kao podatkovno proširenje) uistinu je oduzimalo vrijeme. Za CUP je to obično značilo kombiniranje popisa u Excel-u, a zatim prijenos datoteka putem FTP poslužitelja zbog njihove veličine. Takav postupak zahtijevao je značajno osiguranje kvalitete zbog rizika od ručnih pogrešaka. Nakon što je CUP počeo koristiti okolinu razvijenu u ovom diplomskom radu vrijeme za stvaranja SQL upita se smanjilo s nekoliko sati na nekoliko minuta. „To je bio san za korištenje! Stavio sam štopericu i trebalo mi je 4 minute i 20 sekundi da napravim upit koji kolege obično naprave za sat vremena!“ komentirao je Aisling Miller, viši direktor digitalnog marketinga u CUP-u. Doista, prije bi se CUP često morao oslanjati na pisanje SQL upita kako bi došao do skupova podataka spremnih za promidžbu. Stručnjaci za SQL obično su rijetka pojava za marketinške timove, što dodatno otežava proces segmentacije. Sada, koristeći okolinu razvijenu u ovom diplomskom radu, CUP jednostavno koristi sučelje „*drag and drop*“ za stvaranje ciljanih popisa za izlazne kampanje [40].

5.4.2. Practising Law Institute (PLI)

PLI je nedavno prenio svoje marketinške i segmentacijske aktivnosti s IBM-ove Unica promidžbe na Salesforce Marketing Cloud (SFMC) kao pristupačniju alternativu za tržište njihove, više od 320 000, korisničke mreže koja nastavlja rasti. Tijekom ove tranzicije, početna namjera PLI-a bila je ulagati u obuku za marketinške stručnjake, kako bi mogli stvoriti podatkovna proširenja putem SQL upita u Salesforce Marketing Cloudu. Za neke od njihovih zaposlenika ovo je možda zvučalo kao uzbudljiv izazov, ali za druge kao prilično neprivlačna aktivnost.

Glavni ciljevi PLI su: Demokratizacija podataka i iskorištavanje njihove "zvjezdane sheme" (model podataka usmjeren na kupca koji su interno razvili). Ideja koja stoji iza demokratizacije podataka bila je da marketinški stručnjaci moraju biti u mogućnosti lako segmentirati bez daljnje pomoći svojih kolega iz sektora analizu podataka, kako bi imali tzv. "segmentacijsku samoposlugu". Drugi važan aspekt za PLI bio je iskoristiti shemu zvijezda, što znači da bi marketinški stručnjaci trebali biti u mogućnosti lako povezivati različite skupove podataka koji uključuju aspekte kupaca poput specijalizacije, uloge, kupnje, ponašanje itd., na neprimjetan i lak način. Stoga je opći cilj bio integrirati i implementirati novi pristup automatizaciji marketinga u najkraćem vremenu i na najproduktivniji mogući način.

U potrazi za rješenjima za filtriranje podatkovnih proširenja, PLI je počeo koristiti okolinu opisanu u ovom diplomskom radu kao rješenje koje može uštedjeti vrijeme tijekom izvođenja segmentacije Salesforcea. Nakon što je PLI počeo koristiti navedenu okolinu, počeli su stvarati sve sofisticiranije segmente i to bez oslanjanja na SQL upite [41].

5.5. Dodane funkcionalnosti okoline

Neke su funkcionalnosti dodane nakon pisanja ovog diplomskog rada, a to su: Prilagođene vrijednosti (engl. custom values) koje služe za stvaranje prilagođenih vrijednosti polja koja služe za definiranje klauzule SELECT SQL upita, Dupliciranje s prioritetom (engl. prioritized deduplication) s čim se postiže izdvajanje rezultata s obzirom na definiran prioritet, npr. ako u konačnim rezultatima postoji više različitih vrijednosti za isto polje, ova funkcionalnost filtrira one vrijednosti koje su definirane kao vrijednosti većeg prioriteta [42].

6. ZAKLJUČAK

U ovom diplomskom radu razvijena je okolina za učinkovito stvaranje složenih SQL upita kako bi se riješio problem vezan uz dugotrajnu i neučinkovitu pripremu podataka za marketinšku promidžbu unutar okruženja Salesforce Marketing Cloud. Okolina je potpuno integrirana sa Salesforce Marketing Cloudom bez potrebe za pisanje SQL upita. Pomoću poslužiteljske strane omogućena je komunikacija sa Salesforce Marketing Cloudom te korištenje njihovih resursa poput podatkovnih proširenja i SQL aktivnosti. Poslužiteljska strana sadrži funkcionalnosti pomoću kojih se, uz korištenje korisničkog sučelja, vrlo jednostavno i u vrlom kratkom roku stvara SQL upit. Za razliku od ostalih postojećih rješenja za stvaranje SQL upita, ova okolina je jedina integrirana u Salesforce Marketing Cloudu te pruža najjednostavnije korisničko sučelje za stvaranje SQL upita. Korisniku su omogućene različite funkcionalnosti pomoću kojih se mogu stvoriti različiti jednostavni, ali i složeni SQL upiti. Kroz tri koraka metodom „*drag and drop*“ koriste se funkcionalnosti okoline kako bi se definirali dijelovi za stvaranje SQL upita. Na kraju su opisani neke od povratnih informacija korisnika koji se služe okolinom opisanoj u ovom radu. Provođenjem testova pokazano je kako su povratne informacije pozitivne te potvrđuju ispravnost i učinkovitost okoline. Korištenjem okoline, vrijeme za stvaranje SQL upita skraćeno je što marketinškim stručnjacima donosi više vremena za razvijanje promidžbenih strategija, analizu tržišta i poboljšanje kvalitete marketinških promidžbi.

LITERATURA

[1] What Is a Marketing Campaign? + How to Manage Them Like a Pro, Celine Roque 30. srpnja, 2018. godine, https://business.tutsplus.com/hr/tutorials/what-is-a-marketing-campaign--cms-31524, pristupljeno: 05.05.2020.

[2] How to Effectively Segment Your Data, Michael Linthorst 8. ožuljka, 2013., https://econsultancy.com/how-to-effectively-segment-your-data/, pristupljeno 05.05.2020.

[3] 7 Segmentation Mistakes That Are Costing Your Business Money, Sreeram Sreenivasan, https://www.singlegrain.com/marketing-strategy/7-segmentation-mistakes-that-will-cost-your-business-money/, pristupljeno: 06.05.2020.

[4] Complete Overview of Salesforce Marketing Cloud Studios: and how each one fuels your B2B and B2B2C marketing campaigns, Salesforce, 20. svibnja, 2019.,https://www.saleswingsapp.com/salesforce/overview-of-salesforce-marketing-cloud-studios-components/, pristupljeno: 10.05.2020.

[5] B2B vs B2C marketing – znate li koje su razlike?, GoDigital, https://godigital.hrvatskitelekom.hr/b2b-vs-b2c-marketing-znate-li-koje-su-razlike/, pristupljeno: 11.05.2020.

[6] 10 Mistakes Digital Marketers Make with Salesforce Marketing Cloud, C. Bullock, M. Pollard, https://www.dummies.com/business/marketing/10-mistakes-digital-marketers-make-salesforce-marketing-cloud/, pristupljeno: 17.06.2020.

[7] How to Create a Filtered Data Extension in Salesforce Marketing Cloud, Anthony Lamot, 27. svibnja, 2020., https://www.salesforceben.com/the-drip/how-to-create-a-filtered-data-extension-in-salesforce-marketing-cloud/, pristupljeno: 20.06.2020.

[8] P. Goodey, Salesforce CRM, Packt Publishing, March 2017.

[9] Query Studio for Salesforce Marketing Cloud, Zuzanna Jarczynska, 8. svibnja, 2019., https://sfmarketing.cloud/2019/08/05/query-studio-for-marketing-cloud/, pristupljeno: 21.06.2020.

[10] Meet Audience Studio, Salesforce, https://www.salesforce.com/products/marketing-cloud/data-management/, pristupljeno: 26.06.2020.

[11] 8 Best SQL Query Builders, Tim Keary, 3. prosinca, 2019., https://www.comparitech.com/net-admin/best-sql-query-builders/, pristupljeno: 01.07.2020.

[12] A. E. Nascimento, OAuth 2.0 Cookbook: Protect Your Web Applications Using Spring Security, Packt Publishing, October 2017.

[13] L. Fernández, S. Robles, A. Fortier, S. Ducasse, G. H Rossi, S. E. Gordillo, Meteoroid towards a real MVC for the web, IWST '09: Proceedings of the International Workshop on Smalltalk Technologies, August 2009, pp. 28-37.

[14] SQL Joins, https://www.w3schools.com/sql/sql_join.asp, pristupljeno: 05.07.2020.

[15] Getting Started, https://code.visualstudio.com/docs, pristupljeno: 15.07.2020.

[16] S.Jung, Web Development with Node.js, Journal of Computing Sciences in Colleges, June 2018.

[17] About Node.js, https://nodejs.org/en/about/, pristupljeno: 16.07.2020.

[18] V. Abramova, J. R. Bernardino, NoSQL Databases: MongoDB vs Cassandra, C3S2E '13: Proceedings of the International C* Conference on Computer Science and Software Engineering, July 2013, pp. 14–22.

[19] Testing with Ngrok, https://developer.nexmo.com/tools/ngrok, pristupljeno: 25.07.2020.

[20] What Is Ngrok?, https://www.pubnub.com/learn/glossary/what-is-ngrok/, pristupljeno: 25.07.2020.

[21] S. Milind Dol Aher, D. Gandhmal, Use of 'Basic SQL-The Online Beginner's Guide' Site to Give Hands on Experience of SQL to Students, 2018 IEEE Ninth International Conference on Technology for Education (T4E), December 2018.

[22] Interact with Automation Studio, https://developer.salesforce.com/docs/atlas.en-us.noversion.mc-apis.meta/mc-apis/interacting_with_automation_studio_via_the_web_service_soap_api.htm, pristupljeno: 10.08.2020.

[23] Data Extensions, https://developer.salesforce.com/docs/atlas.en-us.noversion.mc-apis.meta/mc-apis/dataextension.htm, pristupljeno: 10.08.2020.

[24] Create, Retrieve, Update, and Delete Folders, https://developer.salesforce.com/docs/atlas.en-us.noversion.mc-apis.meta/mc-apis/creating_retrieving_updating_and_deleting_folders.htm

[25] Create a Query Activity, https://developer.salesforce.com/docs/atlas.en-us.noversion.mc-apis.meta/mc-apis/creating_a_query_activity.htm

[26] Peform a Query Activity, https://developer.salesforce.com/docs/atlas.en-us.noversion.mc-apis.meta/mc-apis/performing_a_query_activity_using_the_soap_web_service_api.htm

[27] SQL Aliases, https://www.w3schools.com/sql/sql_alias.asp, pristupljeno: 10.08.2020.

[28] FROM Clause Plus JOIN, APPLY, PIVOT (Transact-SQL) https://docs.microsoft.com/en-us/sql/t-sql/queries/from-transact-sql?view=sql-server-ver15, pristupljeno: 12.08.2020.

[29] HTML Introduction, https://www.w3schools.com/html/html_intro.asp, pristupljeno: 25.08.2020.

[30] HTML: HyperText Markup Language, https://developer.mozilla.org/en-US/docs/Web/HTML, pristupljeno: 25.08.2020.

[31] CSS: Cascading Style Sheets, https://developer.mozilla.org/en-US/docs/Web/CSS, pristupljeno: 25.08.2020.

[32] JavaScript Tutorial, https://www.w3schools.com/js/DEFAULT.asp, pristupljeno: 26.08.2020.

[33] JavaScript, https://developer.mozilla.org/en-US/docs/Web/JavaScript, pristupljeno: 26.08.2020.

[34] Tutorial: Intro to React, https://reactjs.org/tutorial/tutorial.html, pristupljeno: 28.08.2020.

[35] React.Component, https://reactjs.org/docs/react-component.html, pristupljeno: 30.08.2020.

[36] Vipul A. M., P. Sonpatki, ReactJS by Example- Building Modern Web Applications with React, April 2016.

[37] National Car Manufacturer and Distributor, https://deselect.io/, pristupljeno: 28.09.2020.

[38] 10 Ways to Create a Great Content Campaign, Kevin Gibbons, 17. siječnja, 2013., https://econsultancy.com/10-ways-to-create-a-great-content-campaign/, pristupljeno: 01.10.2020.

[39] About PLI, https://www.pli.edu/about, pristupljeno: 05.10.2020.

[40] Customer Success Story: Cambridge University Press, https://deselect.io/customer-success-story-cambridge-university-press/, pristupljeno: 06.10.2020.

[41] Customer Success Story: Practising Law Institute (PLI), https://deselect.io/customer-success-story-practising-law-institute-pli/, pristupljeno: 08.10.2020.

[42] C. P. Caldeira, Teaching SQL: a case study, ITiCSE '08: Proceedings of the 13th annual conference on Innovation and technology in computer science education, June 2008.

SAŽETAK

U ovom radu razvijena je okolina za učinkovito stvaranje složenih SQL upita. Okolina rješava problem dugotrajnog pisanja SQL upita prilikom pripreme podataka za marketinške promidžbe unutar Salesforce Marketing Clouda. Korisnik pomoću korisničkog sučelja jednostavnom metodom „*drag and drop*“ prolazi kroz potrebne korake pri stvaranju SQL upita. Na početku je potrebno definirati izvorišna podatkovna proširenja te filtre. Sljedeći korak je odabir ciljnog podatkovnog proširenja u koje će se upisati podaci dobiveni pokretanjem SQL upita. Na kraju se dobiveni podaci prikazuju krajnjem korisniku u obliku tablice. Analizom rada okoline utvrđeno je učinkovitost okoline, te smanjenje vremena stvaranja složenih SQL upita kao i nestanak potrebe za pisanjem SQL upita što pridonosi poboljšanju kvalitete marketinških promidžbi.

Ključne riječi: Salesforce Marketing Cloud, segmentacija, SQL upiti, web aplikacija.

ABSTRACT

Title: Environmental development for effective creation of a complex sql query system

This paper elaborates on developing an environment for efficiently creating complex SQL queries. The problem of resolving the lengthy writing of SQL queries when preparing data for marketing campaigns within the Salesforce Marketing Cloud. Using a simple drag-and-drop user interface, the user goes through the necessary steps in creating an SQL query. Initially, it is necessary to define the source data extension and the filters. The next step is to select the target data extension in which the dana, obtained by running the SQL query, will be saved. Finally, the obtained data is displayed to the end user in the form of a table. Application performance analysis determined the effectiveness of the application in reducing the time of creating complex SQL queries as well as the disappearance of the need to write SQL queries, which contributed to improving the quality of marketing campaigns.

Key words: Salesforce Marketing Cloud, segmentation, SQL queries, web application,

ŽIVOTOPIS

Svetozar Radić rođen je 05. siječnja 1997. u Novom Sadu. Pohađao je osnovnu školu u Silašu te nakon četvrtog razred kreće u osnovnu školu Tenja. III. gimnaziju u Osijeku upisuje 2011. godine gdje razrede prolazi s vrlo dobrim uspjehom. Nakon završetka srednjoškolskog obrazovanja, 2015. godine upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Preddiplomski studij računarstva završava 2018. godine te dobiva titulu inženjera računarstva. Diplomski studij upisuje 2018. na kojem trenutno studira.

PRILOZI

Prilog 1: Dokument rada

Prilog 2: Pdf rada

Prilog 3: Programski kod okoline.